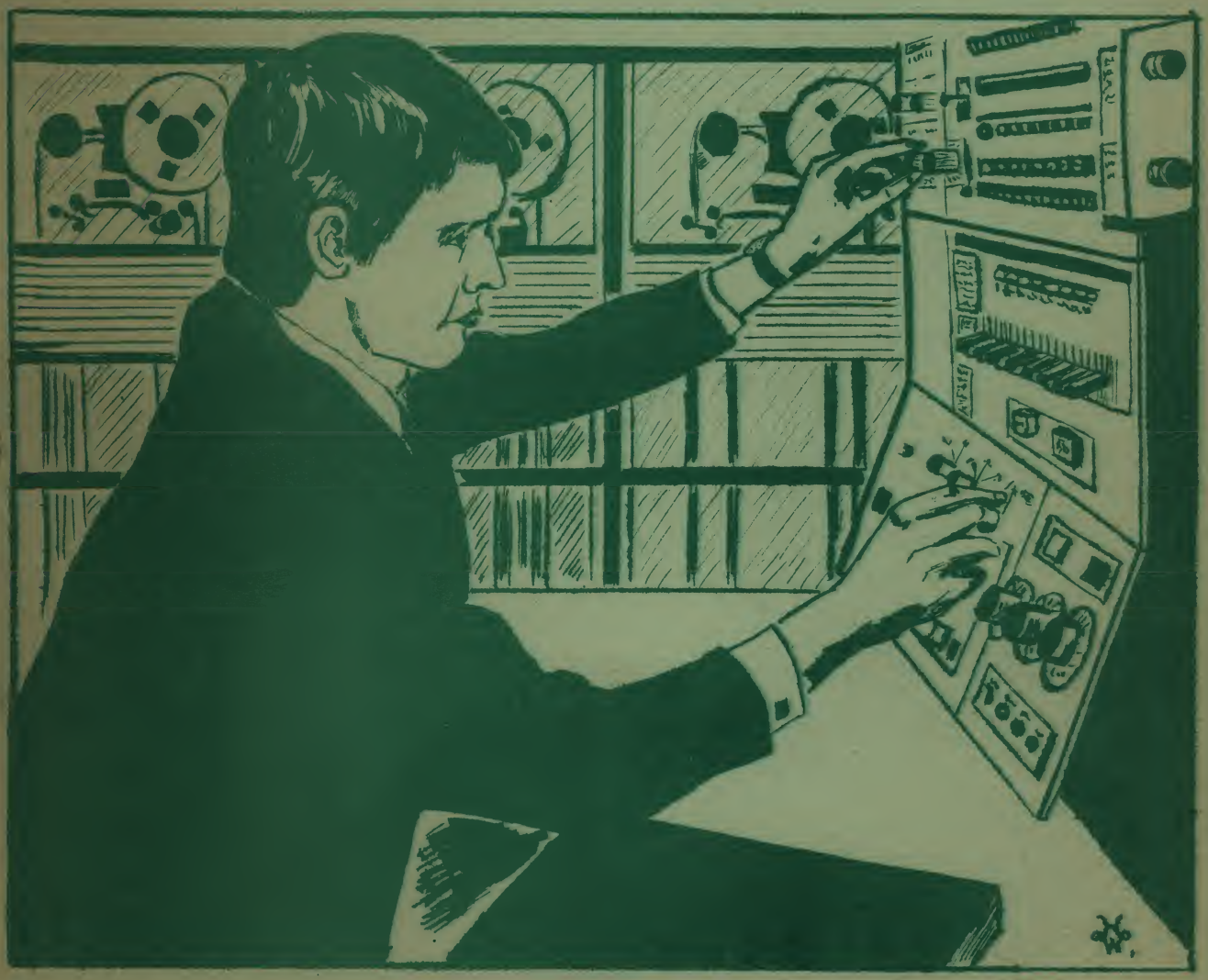
E C S

COMPUTER PROGRAMMEUR



Les 1

Indien wij ons op het terrein van de computer en zijn mogelijkheden begeven, is het van belang iets meer te weten over de voorgeschiedenis die er toe leidde dat wij nu het jaar 2000 tegemoet gaan in een wereld waarin alles geautomatiseerd schijnt te zijn.

Dat de mens altijd naar hulpmiddelen heeft gezocht is een vaststaand feit. Wij denken slechts aan de Chinezen die reeds 2000 jaar voor Christus een telraam gebruikten als hulpmiddel bij het rekenen. Overigens was dit een zeer goede vondst want tot rond het jaar 1600 bleef het telraam het enigste hulpmiddel bij het rekenen, terwijl ook nu nog onze kinderen zich vaak de eerste beginselen van het rekenen eigen maken met behulp van een telraam.

Het is in de 17e eeuw geweest dat men zich meer ging bezig houden met de ontwikkeling van de rekenmachines. Hoewel het moeilijk is vast te stellen welk apparaat nu het eerste de benaming " rekenmachine " kon dragen, neemt men toch vrij algemeen aan dat deze in 1632 is ontworpen door Wilhelm Schickhard, professor in de wiskunde aan de universiteit van Tübingen.

Wat men wel zeker weet is dat Blaise Pascal uit Frankrijk in 1642 een telmachine vervaardigde die kon optellen en aftrekken. Deze machine werkte met van cijfers voorziene tandwielen. Wanneer nu een tandwiel een volledige omwenteling had gemaakt had het daarnaast liggende tandwiel een tiende omwenteling gemaakt. Ofschoon Pascal trachtte zijn ontwerp in serie te laten vervaardigen, moest hij er toch mee staken daar de onderdelen niet geheel nauwkeurig werkten.

Pascal had echter opvolgers. Zo noemen wij Leibniz die in 1671 een machine vervaardigde waarmee het ook mogelijk was te vermenigvuldigen en te delen d.m.v. herhaald optellen en aftrekken. Ook hij strandde echter op de moeilijkheid dat de onderdelen niet voldoende nauwkeurig waren.

Na nog wat mislukte pogingen was het in 1820 de Fransman Thomas de Colman die een rekenmachine vervaardigde die ook de mogelijkheden van vermenigvuldigen en delen in zich had. Mede door de verbeterde technieken was hij in staat om nu machines in serie te laten vervaardigen.

De jaren tussen 1820 en 1900 kenmerkten zich door allerlei technische verbeteringen die er toe leidden dat rond de eeuwwisseling de rekenmachine al echt een handelsartikel was geworden.

Een van de grote ontdekkingen in de 19e eeuw kwam echter van Charles Babbage, een Engelsman die rond 1840 een rekenmachine construeerde met een geheugen. Hierdoor was hij in staat de

machine een kleine reeks instrukties te laten uitvoeren. Babbage maakte gebruik van de uitvinding die Jacquart vóór hem had gedaan, n.l. de ponskaart. Het was Jacquart die de ponskaart had ontdekt als hulpmiddel om patronen mechanisch in een weefsel aan te brengen. Deze kennis gebruikte Babbage om met behulp van gaatjes voorziene kaarten een " programma " in te voeren in zijn rekenmachine.

De belangrijkste stoot in de richting van mechanisatie en automatisering van rekenwerkzaamheden kwam van dr. Herman Hollerith. Deze was omstreeks 1880 direkteur van het amerikaanse Bureau of Census. Eens in de 10 jaar had dit buro als taak de volkstelling te organiseren. De verwerking van de gegevens tot resultaten duurde echter zeven jaar. Ook hij kwam toen op het idee gebruik te maken van Jaquard's uitvinding. Door het aanbrengen van gaatjes in kartonnen kaarten (de latere ponskaarten) was het hem mogelijk geworden om via mechanische weg de kaarten te tellen. Hiertoe ontwierp hij een ponskaartkode en vervaardigde een machine die met stalen pennen de gaatjes in de kaarten kon aftasten. In de machine was een blok met verende pennen aangebracht die op de plaatsen waar een gaatje zat door de kaart zakten en kontakt maakten met een bakje kwik. Doordat het kwik onder elektrische spanning stond, werd hierdoor een stroomcircuit gesloten. Aan iedere pen was een telwerk gekoppeld dat op deze wijze in werking werd gesteld. Door gebruik te maken van zijn vinding slaagde dr. Hollerith er in binnen drie en een half jaar de resultaten van de telling te leveren, wat in die tijd natuurlijk een enorme vooruitgang betekende.

Het is de ponskaart die er verder in belangrijke mate toe bijgedragen heeft dat wij in de 20e eeuw zijn gaan spreken over " conventionele ponskaartenapparatuur " en " computers ". Begrippen waar wij in een volgend hoofdstuk op terug zullen komen.

INFORMATIEVERWERKING.

Indien we over informatieverwerking spreken zijn er drie begrippen die daar een grote rol bij spelen n.l.
a. informatie
b. informatiedrager
c. informatieverwerking

Wat houden nu deze begrippen in? We zullen U dat duidelijk maken.

INFORMATIE

Is datgene wat men nodig heeft om een conclusie uit te trek-

ken of om een beslissing te nemen. Een direktie bijv. zal over veel informatie moeten beschikken om beleidsbeslissingen te kunnen nemen.

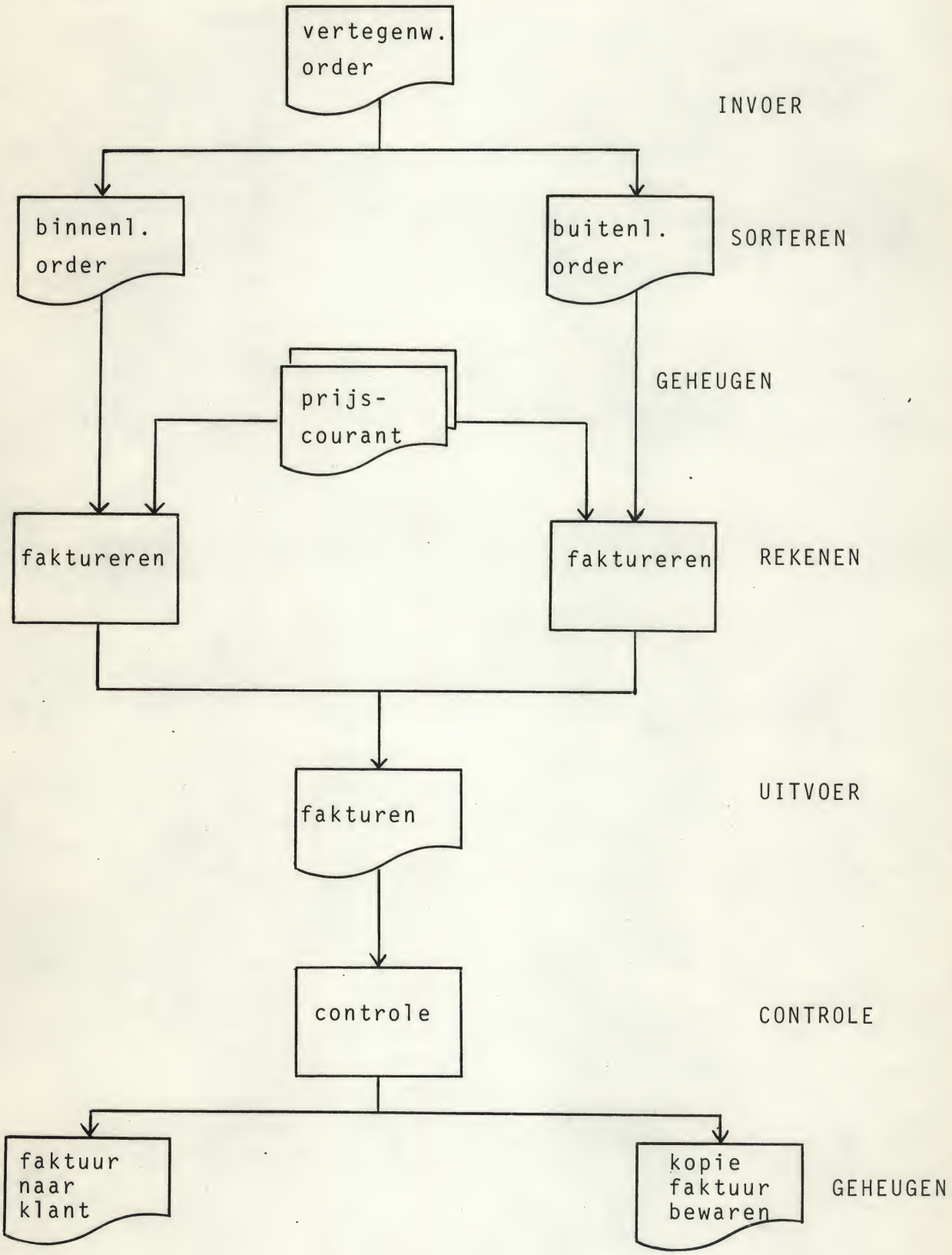
INFORMATIEDRAGER.
Hieronder kunnen we alles verstaan wat informatie bevat. Als voorbeelden zijn hier te noemen : de krant, Uw kursusboek, een kaartenbestand, een grafiek enz. En in de computerwereld is dat bijv. de ponskaart, de magneetband en magnetische schijf.

INFORMATIEVERWERKING.
Hiertoe hebben wij uiteraard gegevens nodig. De ontvangen gegevens (*invoer*) worden gelezen en verwerkt tot resultaten. De resultaten noemt men de *uitvoer*.

Wat houdt nu die verwerking in ?
Over het algemeen kan men stellen dat de informatiewerkzaamheden uit een aantal computer delen kunnen bestaan n.l. :
- het lezen (invoer)
- sorteren, selecteren, (splitsen) rangschikken
- rekenen (verwerken van invoergegevens)
- onthouden (geheugenfunctie)
- controle
- schrijven (resultaten zichtbaar maken = uitvoer)

DOEL VAN DE INFORMATIEVERWERKING.
Alvorens bovenvermelde bewerkingen aan een nadere beschouwing te onderwerpen, zullen wij eerst het doel van de informatieverwerking omschrijven.
Door informatieverwerking tracht men de juiste persoon, op het gewenste moment, over de juiste informatie te laten beschikken.

Aan de hand van het schema op de volgende pagina beschrijven wij nu de meest voorkomende handelingen van een informatieverwerking.
1. Bij een groothandel ontvangt men via een vertegenwoordiger een orderbon. Dit kan men beschouwen als invoer.Men zal deze bon moeten lezen, om
2. de bon te kunnen splitsen naar binnenlandse en buitenlandse afnemers
3. mede door een andere munteenheid en taalgebruik worden de fakturen apart vervaardigd. Tijdens het opstellen hiervan, komt de rekenfunctie naar voren, verder wordt hier nog een ander invoergegeven verwerkt n.l. :
4. de prijscourant. Deze prijscourant heeft in dit voorbeeld de functie van geheugen. Ook het "kladblaadje " wat misschien gebruikt wordt voor het maken van berekeningen kan men als geheugen beschouwen.
5. Het uitschrijven van de faktuur kan men interpreteren als zijnde de uitvoerfunctie n.l. hier wordt het eind-


vertegenw. order
INVOER
binnenl. order
buitenl. order
SORTEREN
prijs-courant
GEHEUGEN
faktureren
faktureren
REKENEN
fakturen
UITVOER
controle
CONTROLE
faktuur naar klant
kopie faktuur bewaren
GEHEUGEN

resultaat bekend gemaakt.

6. Rest ons nog de controlefunctie. Teneinde een foutloze faktuur te versturen, is het noodzakelijk dat men de vervaardigde faktuur eerst controleerd, alvorens ze wordt verzonden.
7. Als laatste handeling hebben we nog een geheugenfunctie n.l. het opbergen van de duplikaat-faktuur. Deze faktuur kan later dienst doen als naslag gegeven.

SAMENVATTING.

Uit bovenstaand voorbeeld, wat overigens zeer beknopt is gehouden, komen toch de belangrijkste aspekten duidelijk naar voren. Wij zullen ze in het kort nog eens herhalen.

INVOER.

Invoer bestaat uit die gegevens welk een informatieverwerkend systeem nodig heeft om te kunnen " verwerken " teneinde een resultaat te kunnen leveren. In ons voorbeeld wordt de invoer geleverd door de vertegenwoordiger, n.l. in de vorm van een orderbon.

SORTEREN.

Sorteren is het rangschikken in een bepaalde volgorde. Deze volgorde kan zowel opklimmend als afdalend zijn.

SELECTEREN.

Selecteren is het uit een groot aantal gegevens afzonderen van bepaalde gegevens. Bijv. alle ordernummers die beginnen met het cijfer 9.

SPLITSEN.

Splitsen is het opdelen van een aantal (bijv) bonnen in één of meerdere eenheden. Het splitsen naar binnenlandse orders en buitenlandse orders bepaalt in ons voorbeeld de splitsing.

VERWERKING.

Deze bestaat uit een aantal opeenvolgende bewerkingen die de gegevens moeten ondergaan alvorens tot een oplossing van het betreffende vraagstuk te komen.In het besproken voorbeeld kan de berekening van de faktuur als een verwerking worden gezien.

OPBERGEN. (geheugenfunctie)

Onder het opbergen verstaat men het vastleggen van gegevens en/of resultaten, op een dusdanige wijze dat zij te allen tijde gemakkelijk zijn te raad plegen.Bijv. het opbergen van de duplikaat-faktuur, of het gebruik van de genoemde prijscourant.

CONTROLE.

Deze functie dient deel uit te maken van ieder informatieverwerkend systeem. Niemand kan instaan voor de juistheid

van gegevens als er vooraf geen waterdichte controle op uitgeoefend is. In een informatieverwerkend systeem heeft controle ook een andere betekenis en wel die van " besturing " M.a.w. het bepalen van de volgorde waarin de diverse bewerkingen verricht dienen te worden.
In het bedrijfsleven is het de afdelingschef die zich met deze controle-functie bezig houdt.

UITVOER.
De uitvoer bestaat uit eindresultaten of tussenresultaten, verkregen na de verwerking van de invoergegevens. Deze kunnen bestaan uit rapporten, statistieken, een bijgehouden voorraadkaart, enz. In ons voorbeeld is de faktuur het uitvoerdocument.

TESTVRAGEN.

1. Omschrijf de begrippen : a. informatie
   b. informatiedrager
   c. informatieverwerking
2. Wat is sorteren ?
3. Wat is selecteren ?
4. Wat is splitsen ?
5. Bedenk enkele geheugenvormen (wij hebben reeds de prijscourant genoemd)

Automatisering in 1915 en anno 1971.

OPBOUW VAN EEN REKENMACHINE

Prof. ir. D.H. Wolbers

Een publikatie van de Stichting Het Nederlands Studie-Centrum voor Informatica, Amsterdam.

1. INLEIDING

Elektronische rekenmachines worden tegenwoordig zeer veel gebruikt als hulpmiddelen bij het oplossen van problemen, zowel van administratieve als wetenschappelijke aard. Daarbij moet de nadruk worden gelegd op het woord hulpmiddel. Een elektronische rekenmachine is slechts in staat zeer snel en accuraat een aantal voorgeschreven berekeningen uit te voeren.
Voor ieder nieuw probleem moet men steeds volledig en in alle details aan de machine voorschrijven welke berekeningen uitgevoerd moeten worden. Bovendien moet dit voorschrift of programma nog in een speciale codevorm worden opgezet. Deze werkzaamheden noemt men programmeren.
Soms wordt al het voorbereidende werk door een persoon of een groep mensen gedaan.
Voor zeer grote en vooral administratieve problemen wordt het werk verdeeld over een aantal verschillende groepen.
Het was veelal gebruikelijk daarbij te onderscheiden :

- *VOORONDERZOEKERS OF SYSTEEMANALISTEN*

Zij gaan bij een gegeven probleem na, welke informatie noodzakelijk is voor de oplossing hoe en waar deze is te verkrijgen en op welke wijze deze informatie in grote lijnen verwerkt moet worden. De resultaten legt men vast in overzichtelijke schema's, stroomschema's of stroomdiagrammen genoemd. De specifieke kennis van de rekenmachine speelt slechts een kleine rol. De kennis van de systeemanalisten is in de eerste plaats kennis van het probleem.

- *PROGRAMMEURS*

Zij verfijnen het gegeven stroomschema, daarbij reeds rekening houdende met speciale machine-eigenschappen.

- *CODEURS*

Zij zorgen voor omzetting van het verfijnde stroomschema in gecodeerde instructies.

De laatste jaren is door de ontwikkeling van de programmeringstechniek gepaard gaande met een verdere technische ontwikkeling van de rekenmachine vooral het onderscheid tussen de laatste 2 groepen praktisch verdwenen.
Voor de ontwikkeling van de z.g. programmeertalen kan de systeemanalist zijn werk gedeeltelijk vaak reeds zo opleveren dat het direct verwerkt kan worden, m.a.w. hij heeft dan zelfs het werk tot en met de co-

deur overgenomen. Daartegenover moet de programmeur nu vaak nog een deel doen van wat vroeger door de systeemanalist werd uitgevoerd en daarnaast goed op de hoogte zijn van de moderne programmeringstechnieken om die dan te kunnen toepassen.
Het doel van deze kursus is nu speciaal gericht op de taak van deze moderne programmeur. Dat betekent een behoorlijk inzicht in de principiële werking van de rekenmachine. Een goed begrip voor de verschillende niveau's waarop geprogrameerd kan worden en hoe deze niveau's kunnen samenwerken. Daarnaast een zekere vaardigheid verkrijgen in programmeren op basisniveau onder toepassing van de elementaire programmeringstechnieken zoals die in de loop der jaren ontwikkeld zijn.
De daarvoor vereiste oefening is alleen te verkrijgen indien men waar nodig dan uitgaat van een bepaalde rekenmachine. Voor deze kursus is dat de SERA *(STICHTING'S EENVOUDIGE REKEN AUTOMAAT)*. Het is in feite een hypothetische machine, zoals die heden ten dage gebruikt wordt. Verder is de SERA een voorbeeld van een *d i g i t a l e* rekenmachine. Naast digitale rekenmachines kent men nog *a n a l o g e* rekenmachines. Een analoge rekenmachine werkt met continue, meestal, physische grootheden en berust op het principe van meten. Inherent bij deze machines is de beperkte nauwkeurigheid. Voor bepaalde berekeningen, zoals oplossen van differentiaalvergelijkingen kan de analoge rekenmachine echter veel sneller resultaten geven dan een digitale. Deze machines worden dan ook uitsluitend gebruikt voor wetenschappelijke berekeningen. Een digitale rekenmachine werkt met discrete waarden, d.w.z. met aantallen en berust op het principe van tellen. In principe kan men met een digitale rekenmachine werken met iedere gewenste nauwkeurigheid.

Voorbeelden van eenvoudige analoge en digitale apparaten zijn snelheidsmeter en kilometerteller in een auto. De snelheidsmeter is daarbij een analoog apparaat, waarbij op ieder moment een hoekverdraaiing van een wijzer evenredig is met de snelheid, dus werkt onmiddellijk en met beperkte nauwkeurigheid. De kilometerteller is digitaal, hij verspringt in discrete eenheden waarbij het aantal cijfers slechts afhankelijk is van de technische uitvoering.
Voor administratieve doeleinden wordt slechts gebruik gemaakt van digitale rekenmachines en deze kursus handelt verder dan ook uitsluitend over het programmeren voor digitale machines en daarvan in het bijzonder de SERA.

2. OPBOUW VAN EEN REKENMACHINE.

In het algemeen is een rekenmachine samengesteld uit 5 grote delen, die schematisch zijn aangegeven in fig. 1.

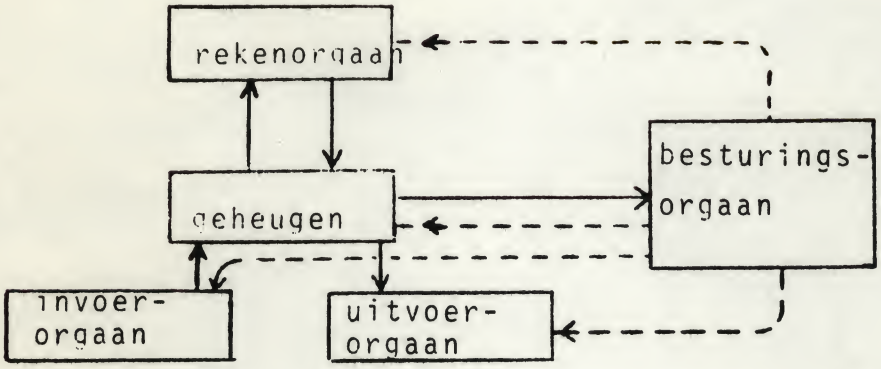


fig. 1.

Om de functies van deze onderdelen te begrijpen kunnen we het beste de rekenmachine vergelijken met een mens, die met behulp van een tafelrekenmachine, papier en potlood een grote berekening uitvoert. De tafelrekenmachine dient dan alleen om de optellingen, aftrekkingen eventueel vermenigvuldigingen en delingen uit te voeren. In de elektronische rekenmachine wordt deze functie vervuld door het rekenorgaan. Het papier en potlood wordt gebruikt om de getallen, waarmee gerekend moet worden, alsmede de resultaten neer te schrijven. Hiermee komt overeen het geheugen. Tussen geheugen en rekenorgaan bestaan verbindingen, zodat getallen uit het geheugen naar het rekenorgaan gebracht kunnen worden en omgekeerd berekende resultaten vanuit het rekenorgaan weer terug naar het geheugen.
Als papier gebruiken we bij voorkeur gelinieerd papier met een zeker aantal regels per blad, terwijl we op iedere regel een beperkt aantal cijfers kunnen schrijven. In de machine komt het geheugen overeen met dit papier. In plaats van regels spreken we hier over WOORDEN. Voor de SERA machine is dit aantal 10.000 woorden. Ieder woord kan daarbij een getal bevatten van maximaal 11 cijfers benevens een + of - teken.

ADRES
Ten einde de weg te kunnen vinden in het geheugen gaan we deze woorden nummeren. Zo'n nummer noemen we het *adres* van een woord. Deze nummering begint bij 0 en loopt tot en met 9999. Door bij 0 te beginnen bestaat het adres altijd uit 4 cijfers. Een adres heeft op zichzelf dus niets te maken met het getal, dat op die plaats in het geheugen staat.

Wanneer in het woord met adres 100 het getal +3785936 staat en we willen dit getal naar het rekenorgaan brengen dan kunnen we dit omschrijven met de opdracht " breng de inhoud van 100 naar het rekenorgaan "
Symbolisch geven we dit aan met
(100) ——→ rekenorgaan.
De ronde haakjes worden daarbij gebruikt voor het begrip " de inhoud van "
In het gegeven voorbeeld geldt dus

(100) = +3785936
Deze notatie wordt algemeen gebruikt. Zeer vaak treft men voor het omgekeerde geval nog de notatie) (aan. Dit gebruikt men wanneer men het getal, dat in een woord staat, kent maar het adres van dit woord wil aangeven. Voor dit voorbeeld geldt dan
) +3785936 (= 100
In woorden : " het adres van het woord, waar het getal + 3785936 staat, is 100 "

FUNKTIES VAN HET GEHEUGEN
We hebben nu de eerste funktie van het geheugen leren kennen, n.l. als opslagplaats van de getallen, waarmee we willen rekenen.
Voor de tweede funktie van het geheugen denken we weer terug aan de handrekenaar.
Wanneer de totale berekening niet al te ingewikkeld is zal hij alle noodzakelijke manipulaties met de tafelrekenmachine en het neerschrijven van getallen " uit het hoofd doen ". Stellen we ons nu voor een bijzonder ingewikkelde en langdurige berekening, d.w.z. een berekening die in een groot aantal stappen moet worden uitgevoerd. Teneinde dan vergissingen te voorkomen noteren we vaak van te voren, ook op papier, welke berekeningen we stap voor stap zullen moeten uitvoeren.
Aan de hand van dit gaan we dan de berekening uitvoeren.
In vele gevallen zal de persoon, die de berekening uitvoert zelfs een andere zijn dan degene, die het berekeningsschema opstelt. Nu weer terug naar de rekenmachine.
Als papier voor het berekeningsschema gebruiken we het geheugen. Daarom kan een woord uit het geheugen in plaats van een getal ook een berekeningsopdracht bevatten. Met de man, die de berekening uitvoert, komt dan overeen het *BESTURINGSORGAAN*. Vandaar de gestippelde lijnen vanaf dit orgaan, die dus de betekenis van besturingslijnen hebben. Tussen geheugen en besturingsorgaan is echter ook een getrokken lijn in de richting n a a r besturingsorgaan en hiermee krijgt het besturingsorgaan zijn opdrachten uit het geheugen, overeenkomstig het rekenschema. Tenslotte de man die het rekenschema opstelt. Dit blijft een mens, de programmeur, die de machine moet zeggen wat gedaan moet worden.

Hierna blijven nog over de in- en uitvoerorganen. We moeten ons realiseren, dat de drie eerder genoemde delen, n.l. rekenorgaan, geheugen en besturingsorgaan technisch gemaakt zijn als elektronische schakelingen. Getallen komen dus bijvoorbeeld voor in de vorm van elektrische stromen of spanningen of door plaatselijke magnetisatie van materialen. In deze vorm is dit voor ons mensen niet direkt waarneembaar en kunnen wij er niet mee werken. Daarom zijn er twee communicatiekanalen gemaakt. Ten eerste het invoerorgaan. Hieraan kunnen wij mense-

lijke informatie toevoeren. In de meeste gevallen vindt dit plaats in de vorm van ponskaarten. Daarbij worden de gegevens, die we aan de machine willen geven, vastgelegd in de vorm van geponste gaatjes in ponskaarten. Deze kaarten worden dan *"gelezen"* door een ponskaartenlezer, d.w.z. de kaarten worden mechanisch afgetast op de aanwezigheid van gaatjes en dit wordt omgezet in elektrische impulsjes, die worden doorgegeven aan het geheugen. Nadere gegevens over de indeling en het gebruik van ponskaarten worden later verstrekt. Voor administratieve problemen worden hoofdzakelijk ponskaarten gebruikt. We willen er hier echter op wijzen dat in het algemeen naast ponskaarten ook wel gebruik gemaakt wordt van ponsband.

PONSBAND.
Is papier van enkele cm. breedte en kan 250 tot 300 m lang zijn. In de breedterichting kan men een aantal gaatjes naast elkaar ponsen. Dit noemt men wel een *REGEL* of een *SYMBOOL* of een *KARAKTER*.

4 SOORTEN PONSBAND.
Men kent 4 soorten ponsband, n.l. met 5,6,7 en 8 gaatjes per regel. Men spreekt dan van 5-, 6-, 7- of 8-kanaalsponsband. Bij ieder type is dan bovendien per regel nog een extra gat aanwezig, iets kleiner dan de overige, dat *transportgat* genoemd wordt. Zoals de naam aangeeft dient dit alleen voor de voortbeweging van de band. Een ponsband bevat ongeveer 400 regels per meter. Per regel kunnen we bepaalde gaatjes wel en andere niet ponsen, terwijl we daarbij aan iedere combinatie een bepaalde betekenis toekennen. Voor 7-kanaalsband betekent dit dat we $2^7 = 128$ mogelijkheden hebben. Hiermee kunnen we dan bijv. de 10 cijfers 0 t/m 9 alsmede de 26 kleine letters en de 26 hoofdletters van het alfabet voorstellen. Verder hebben we nog combinaties voor leestekens, guldentekens enz. Het komt er dus op neer, dat onze menselijke schrijftekens vertaald worden in combinaties van gaatjes in deze ponsband. Ditzelfde ontmoeten we straks bij ponskaarten.

Het "vertalen" vindt plaats geheel los van de elektronische rekenmachine met behulp van speciale ponsmachines. Dit is echter geheel handwerk, waarbij symbool voor symbool met toetsen moet worden aangeslagen,soms in een vorm zoals bij een schrijfmachine, soms door speciale toetsencombinaties. Is dit ponswerk eenmaal gedaan, dan kan het inlezen in de rekenmachine betrekkelijk snel plaats vinden. Moderne ponsbanden-lezers kunnen ponsband lezen met een snelheid van 1000 regels per seconde. Dit ponswerk kan bij automatiseringsproblemen met massale hoeveelheid informatie, die ingevoerd moet worden, vaak het grootste struikelblok vormen. Men is daarom al jaren bezig met de ontwikkeling

van apparaten, die normaal menselijk leesbaar schrift ook machinaal kunnen lezen. Pas de laatste tijd komen ook enkele apparaten op de markt, die inderdaad deze mogelijkheid bieden. In de meeste gevallen nog wel beperkt tot gedrukt schrift, dat bovendien nog voldoet aan bepaalde eisen en verder in een enkel geval ook geschreven schrift mits dat ook voldoet aan bepaalde minimum voorwaarden. Toch kan men nog niet spreken van een algemeen gebruik van dergelijke apparaten en de grote massa van invoerapparaten wordt nog steeds gevormd door de ponskaartenlezer.

Tenslotte moeten nog genoemd worden magneetbanden als invoer. Het gebruik hiervan wordt besproken in een apart hoofdstuk, waaruit tevens zal blijken, dat magneetbanden eigenlijk een vorm van geheugenuitvoering zijn.

Het tweede communikatiekanaal tussen machine en mens wordt gevormd door het uitvoerorgaan. Hierbij vinden we allereerst dezelfde systemen als bij invoer, n.l. ponskaarten en ponsband. Daarbij worden door het uitvoerorgaan gaatjes in het papier geponst overeenkomstig de informatie die in het geheugen aanwezig is. Het mechanische werk dat gedaan moet worden bij het ponsen van deze gaatjes is oorzaak dat het ponsen van ponskaarten en ponsband langzamer gaat dan het overeenkomstige lezen. Voor ponsband geldt normaal 100 regels per sekonde. Een zeer snelle en gecompliceerde ponsmachine komt toch niet verder dan 300 regels per sekonde.

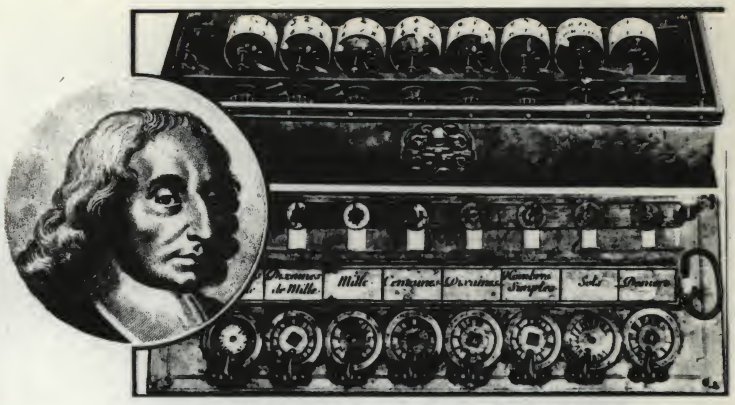
SNELDRUKKER
Bij de uitvoer heeft men echter nog een belangrijke mogelijkheid namelijk rechtstreeks in gedrukt schrift. Eenvoudig en betrekkelijk langzaam is dat in de vorm van een elektrische schrijfmachine ; snel en in massale hoeveelheden door middel van een zgn. " sneldrukker ". Deze kan ongeveer 1000 regels/min. afdrukken met 120 tot 160 tekens/regel. Deze sneldrukker wordt vooral gebruikt in de administratieve techniek, waar het afdrukken van de berekende resultaten vaak plaatsvindt op reeds voorbedrukte formulieren.

TECHNISCHE UITVOERING
We willen nu de technische uitvoering van de onderdelen iets nader bekijken, althans zover dit nodig is voor een goed begrip van het gebruik.
Een *digitale* rekenmachine werkt met onderdelen die discrete posities, toestanden of grootheden kunnen innemen of voorstellen. Op zichzelf behoort ook de normale tafelrekenmachine tot dit type machine. De discrete toestanden zijn hier op mechanische wijze gerealiseerd door middel van allerhande tandwieltjes en tandradjes, die mechanisch met elkaar gekoppeld zijn.
Technisch is het eenvoudig een wieltje zo te maken, dat het in een beperkt aantal gefixeerde standen kan staan.

Daarbij geeft het aantal van deze mogelijke standen geen enkele complicatie of dit nu 5 of 10 of 20 is. Wegens ons tientallig stelsel waarmede wij mensen rekenen, komen daardoor getalwieltjes met 10 standen dan ook veel voor. Bij een elektronische digitale rekenmachine zouden wij bij voorkeur ook gebruik maken van elektronische equivalenten van deze mechanische wieltjes. Dit is helaas in direkte vorm niet mogelijk. Wel kan men echter betrekkelijk gemakkelijk en betrouwbaar elektronische schakelingen alsmede elektrische en magnetische materialen fabriceren, die zich ieder voor zich steeds in één van twee stabiele toestanden kunnen bevinden. Hierbij te denken aan al of niet stroom of spanning; positieve of negatieve stroom of spanning, magnetisatie in de ene of andere richting. Dit duale karakter is oorzaak geweest, dat elektronische rekenmachines in principe steeds werken in het *TWEETALLIGE STELSEL*.
Daarbij moet direkt worden opgemerkt dat vele elektronische digitale rekenmachines worden aangegeven als decimale machines, zoals o.a. ook de SERA. Bekijkt men echter deze machines in detail dan blijkt toch weer het tweetallig karakter. Ook bij het programmeren voor deze decimale machines heeft men in enkele gevallen met deze eigenschap te maken. We worden dan geconfronteerd met in het algemeen gesproken het rekenen in een ander talstelsel en vooral het omrekenen van en naar een ander talstelsel vanuit het ons bekende decimale stelsel. In de eerste plaats geldt dat dan voor het binaire stelsel, maar daarmee samenhangend spelen ook vaak het 8 en 16-tallig stelsel een rol. Voor normaal gebruik is dit feit minder belangrijk omdat door ingebouwde hardware opdrachten of anders door standaardsubprogramma's de nodige omrekeningen wel plaats vinden.
In bijzondere gevallen wordt echter de programmeur bij het ontwerpen en testen van bepaalde basisprogramma's toch wel geconfronteerd met de omrekening zelf, en moet soms met de hand enkele voorbeelden doorrekenen.

In het volgende geven we daartoe enkele praktische richtlijnen.
Beschouw het getal 71550 in het 10-tallig stelsel.
We noteren dit als $(71550)_{10}$
de betekenis is eigenlijk $0x1+5x10+5x10^2+1x10^3+7x10^4$
We kunnen dit ook schrijven als $0+10x(5+10x(5+10x(1+10x7)))$
In de laatste vorm is het dus het laatste cijfer 0 plus 10 maal de rest van het getal. Deze rest wordt weer op dezelfde wijze geschreven als 5 plus 10 maal de dan overblijvende rest enz.
Stel we willen dit getal schrijven in het 8-tallig stelsel. Dan willen we eigenlijk zo'n soort voorstelling namelijk het laatste cijfer van het 8-tallig getal plus 8 maal de rest van het getal en dan deze rest ook weer op dezelfde manier enz.
Dit geeft meteen de omrekeningsmethode, namelijk, deel het gegeven getal door 8 en bepaal quotient en rest.

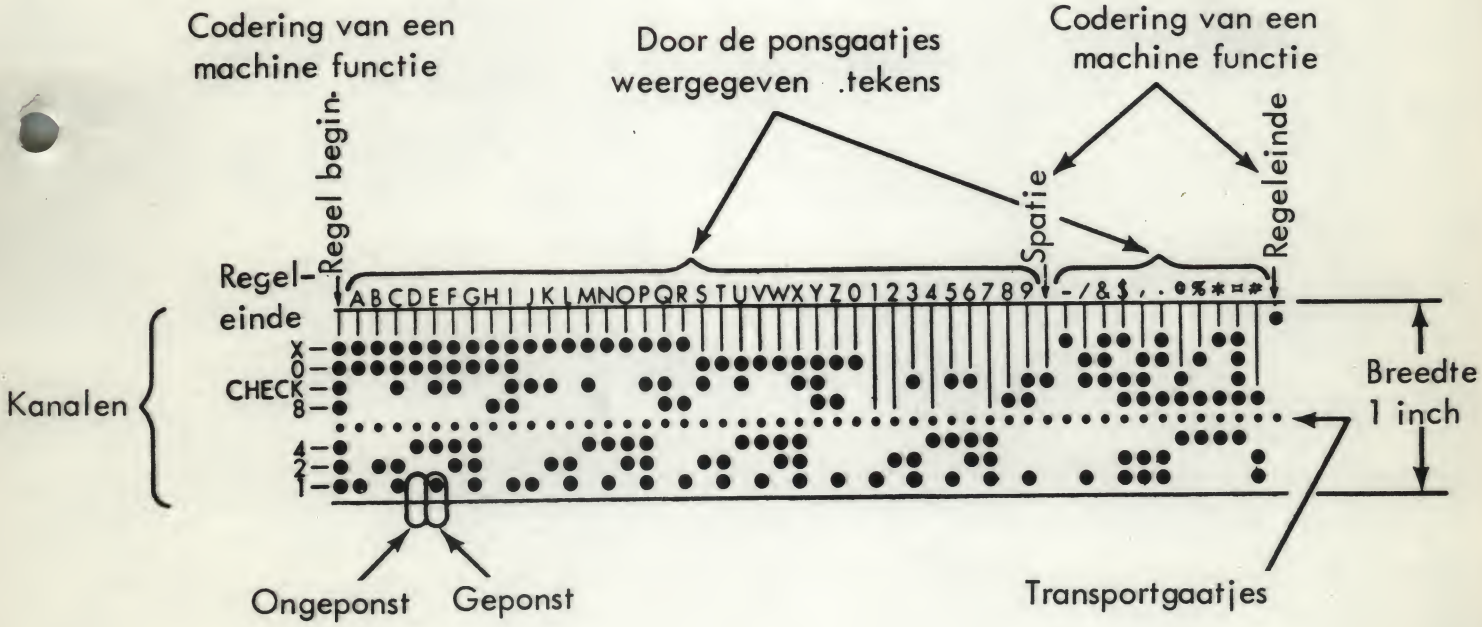
de ponstypiste verricht haar werk met grote nauwkeurigheid



Blaise Pascal met rekenmachine

een moderne computer-installatie

8 KANALEN PAPIEREN PONSBAND



Dus 71550=6+8x8943
Hieruit volgt 6 is laatste cijfer van het 8-tallig getal.
Het overblijvende quotient 8943 behandelen we nu weer op dezelfde manier enz. n.l.
8943=7+8x1117
1117=5+8x139
139=3+8x17
17=1+8x2
2=2+8x0
Het 8-tallig equivalent van $(71550)_{10}$ is dus $(213576)_8$

Terugomzetting naar 10-tallig verloopt precies andersom.
Bijvoorbeeld : $(372156)_8$

3+8x0=3
7+8x3=31
2+8x31=250
1+8x250=2001
5+8x2001=16013
6+8x16013=128110

Dus $(372156)_8=(128110)_{10}$

Een ander voorbeeld $(5BEA4D)_{16}$. Hierin staan de letters A t/m F respectievelijk voor de cijfers 10 t/m 15 dus :

10 = A	13 = D
11 = B	14 = E
12 = C	15 = F

De berekening gaat dan als volgt :

```
                5
               16 x
               30
               50
               80
               11 +
               91
               16 x
              546
              910
             1456
               14 +
             1470
               16 x
             8820
            14700
            23520
               10 +
            23530
               16 x
           141180
           235300
           376480
                4 +
           376484
```

zie volgende pagina.

Ir. D.H. Wolbers

```
   376484
       16 x
  2258904
  3764840
  6023744
       13 +
  6023757
```

Dus $(5BEA4D)_{16}=(6023757)_{10}$

We willen nu het laatste getal weer omzetten in het 8-tallig stelsel.

```
6023757
 752969  5
  94121  1
  11765  1
   1450  5
    183  6
     22  7
      2  6
```

Dus $(6023757)_{10}=(26765115)_8$

Dit laatste hadden we ook vlugger kunnen bereiken door uit te gaan van het oorspronkelijke 16-tallige getal en dit binair te schrijven. Ieder 16-tallig cijfer komt dan overeen met 4 bits. Verdelen we daarna het hele binaire getal in groepjes van 3 bits, dan stelt ieder groepje een 8-tallig cijfer voor :

5BEA4D 0101 1011 1110 1010 0100 1101

Nu in groepjes van 3 van rechts af beginnen :

010	110	111	110	101	001	001	101	geeft
2	6	7	6	5	1	1	5	

OMREKENEN VAN BREUKEN

Een ander probleem bij het omrekenen vormen de breuken. Welke methode men ook volgt, het gaat altijd met vrij veel rekenwerk gepaard. Een vrij overzichtelijke methode, die men bovendien voor zoveel cijfers nauwkeurig kan voortzetten als men wenst is de volgende :
Beschouw de breuk $(0,74392)_{10}$. Vermenigvuldigen we dit getal met 10 dan ontstaat $(7,4392)_{10}$ waardoor het eerste cijfer achter de komma nu juist voor de komma ontstaat. Nemen we $(o,34715)_8$ en vermenigvuldigen we dit met 8 dan ontstaat $(3,4715)_8$

In beide gevallen konden we deze vermenigvuldiging eenvoudig uitvoeren omdat het niets anders betekende dan de cijfers één plaats naar links te schuiven, waardoor één cijfer voor de komma ontstond.
Deze eigenschap kunnen we ook gebruiken voor de omzetting, alleen moeten we in dat geval dan echt vermenigvuldigen.

Ir. D.H. Wolbers

Stel als voorbeeld het omzetten van $(0,74392)_{10}$ in een 8-tallige breuk. We vermenigvuldigen dan deze breuk met 8 dus :
8x0,74392=5,95136
Het cijfer 5 voor de komma betekent nu dat dit het eerste cijfer achter de komma is van de gevraagde 8-tallige breuk. We laten deze 5 weg en gaan met de rest van de breuk gewoon verder,
8x0,95136=7,61088
zodat het tweede cijfer dus wordt 7.
Zo kunnen we zoveel cijfers uitrekenen als we wensen.
Bijvoorbeeld :
0,74392
3,95136
7,61088
4,88704
7,09632
0,77056
6,16448
Dit betekent dat $(0,74392)_{10}=(0,574706)_8$ maar alleen ongeveer !

Het meeste rekenwerk geeft het omzetten van een breuk in een ander talstelsel naar een 10-tallige breuk. Men moet dan steeds met 10 vermenigvuldigen, maar het vervelende is dat dit in feite moet gebeuren met vermenigvuldigtafels die voor het gegeven talstelsel geldig zijn en daaraan zijn wij eenvoudig niet gewend. Met enige moeite kan men nog wel in een ander talstelsel optellen of een getal verdubbelen (bij zichzelf optellen)
Hierop gebaseerd kan men de vermenigvuldiging met 10 uitvoeren door driemaal achtereen te verdubbelen en daarbij het dubbele van het oorspronkelijke getal op te tellen.
Dit berust op 10=2x2x2+2
We nemen dus eigenlijk 8x plus 2x het getal. Bij omzetting van 8-tallig naar 10-tallig heeft men nog het voordeel dat het 8-voud eenvoudig door schuiven te bereiken is.

We nemen als voorbeeld het resultaat van de laatste omrekening $(0,574706)_8$ en rekenen dit terug naar 10-tallig.

0,574706	1x
1,371614	2x
5,74706 +	8x
7,340674	
0,701570	2x
3,40674 +	8x
4,31053	
0,62126	2x
3,1053 +	8x
3,72656	

zie volgende pagina.

Ir. D.H. Wolbers

```
3,72656
I,65534   2x
7,2656    8x
-------- +
9,14314
0,30630   2x
1,4314    8x
-------- +
1,7377
I,6776    2x
7,377     8x
-------- +
9,2766
0,5754    2x
2,766     8x
-------- +
3,5634
```

Dus $(0,574706)_8 = (0,7439193)_{10}$ ongeveer !

We krijgen dus niet meer de oorspronkelijke 10-delige breuk terug. Door meer cijfers achter de breuk in rekening te brengen kan men de overeenkomst dus beter maken, of anders gezegd de onnauwkeurigheid verkleinen.
Hierbij dient echter bedacht te worden dat een breuk die in het ene talstelsel "opgaat" in een ander talstelsel kan leiden tot een repeterende breuk en omgekeerd.
Vergelijk bijvoorbeeld 1/3. In het 10-tallig stelsel voert dit tot $(o,3333.....)_{10}$ soms genoteerd als $(0,\dot{3})_{10}$.
In het 3-tallig stelsel geldt echter $1/3 = (0,1)_3$

Voeren we het laatste voorbeeld nogmaals uit door eerst de 8-tallige breuk om te zetten in 16-tallige breuk via een binaire breuk.

$(0,574706)_8 = 0,101\ \ 111\ \ 100\ \ 111\ \ 000\ \ 110 =$
$0,1011\ \ 1110\ \ 0111\ \ 0001\ \ 1000 =$
$(0,BE718)_{16}$

Nu is het vermenigvuldigen met 8 niet meer door schuiven te bereiken, waardoor de berekening dan als volgt verloopt :

```
0,BE718
1,7CE30   2x
2,F9C60   4x
5,F38C0   8x
--------- +
7,706F
0,EODE    2x
1,C1BC    4x
3,8378    8x
--------- +
4,6456
0,C8AC    2x
1,9158    4x
3,22B0    8x
--------- +
3,EB5C
I,D6B8    2x
3,AD70    4x
7,5AE0    8x
--------- +
9,3198
```

zie volgende pagina

9,3198
0,6330 2x
0,C660 4x
1,8CC0 8x
0,6330 +
1,EFF0
1,DFE 2x
3,BFC 4x
7,7F8 8x
1,DFE +
9,5F6
0,BEC 2x
1,7D8 4x
2,FB0 8x
0,BEC +
3,B9C

Ook nu vinden we dus weer $(0,BE718)_{16}$-$(0,7439193)_{10}$ ongeveer.

HET OMREKENEN VAN EEN GETAL MET CIJFERS VOOR EN ACHTER DE KOMMA.

Tenslotte het geval dat men een willekeurig getal met een aantal cijfers voor en achter de komma moet omrekenen. Bijvoorbeeld gegeven $(367,28)_{10}$ en gevraagd dit getal in 8-tallig stelsel met een nauwkeurigheid van 3 cijfers achter de komma.
Dit vraagstuk valt volledig uiteen in 2 delen n.l. het geheel getal gedeelte en het breukgedeelte, dus afzonderlijk $(367)_{10}$ en $(0,28)_{10}$

Dit verloopt als volgt :

367 | 7
45 | 5
5

0,28
8
2.24 x
8
1,92 x
8
7,36 x
8
2,88 x

En daarmee afgerond op 3 cijfers achter de komma
$(367,28)_{10}=(557,217)_{8}$

HET TWEETALLIG STELSEL

Komen we nu tot het tweetallig stelsel. Dan kent men slechts twee cijfers en wel 0 en 1 Deze komen dan nog overeen met de 0 en 1 in ons tientallig stelsel. Ons getal 2 vereist in het binaire stelsel reeds twee cijfers n.l. 10. Ons getal 3 wordt dan 11. Voor het getal 4 zijn zelfs twee cijfers niet meer toereikend, zodat we krijgen 100.

Hieronder de eerste 32 equivalenten.

10-tal	2-tal	10-tal	2-tal	10-tal	2-tal
o	0	11	1011	22	10110
1	1	12	1100	23	10111
2	10	13	1101	24	11000
3	11	14	1110	25	11001
4	100	15	1111	26	11010
5	101	16	10000	27	11011
6	110	17	10001	28	11100
7	111	18	10010	29	11101
8	1000	19	10011	30	11110
9	1001	20	10100	31	11111
10	1010	21	10101	32	100000

Ons getal 32 vereist dus reeds 6 cijfers in het tweetallig stelsel. Steeds wanneer we in het tientallig stelsel een nieuwe 2 macht bereiken, betekent dit een extra cijfer in het tweetallig stelsel. Het tweetallig stelsel vereist dus een groter aantal cijferplaatsen. Getallen worden daardoor gauw "groot". Voor vergelijk is het beste te onthouden $2^{10} = 1024$. Ronden we 1024 even af op $1000 = 10^3$ dan is dus 2^{10} ongeveer gelijk aan 10^3. D.w.z. een nauwkeurigheid van 3 cijfers in ons tientallig stelsel is ongeveer even groot als 10 cijfers in het tweetallig stelsel.

Het rekenen in dit tweetallig stelsel is bijzonder eenvoudig. Vooral optellen en aftrekken. Enkele voorbeelden met daarbij de decimale voorstelling.

```
  87        1010111
  35         100011
 ---  +    -------  +
 122       1111010

 118        1110110
  45         101101
 ---  +    -------  +
 163       1010001
```

REKENREGELS

De rekenregels voor optellen zijn zeer eenvoudig. Indien geen overdracht van vorige plaats dan geldt :

0 + 1 geeft 1 en geen nieuwe overdracht
1 + 0 geeft 1 en geen nieuwe overdracht
0 + 0 geeft 0 en geen nieuwe overdracht
1 + 1 geeft 0 en wel nieuwe overdracht

Indien er wel een overdracht van vorige plaats is dan geldt

0 + 1 geeft 0 en wel nieuwe overdracht
1 + 0 geeft 0 en wel nieuwe overdracht
0 + 0 geeft 1 en geen nieuwe overdracht
1 + 1 geeft 1 en wel nieuwe overdracht

Soortgelijke regels gelden ook bij aftrekken, waarbij in plaats van overdracht nu het "lenen" een rol speelt,

bijvoorbeeld :

87	1010111	
35 -	100011	-
52	110100	
118	1110110	
45 -	101101	-
73	1001001	

Er zijn machines, die volledig in dit stelsel werken. Daarbij wordt dan gewerkt met een standaard aantal *BITS* Een bit (afkorting van BINARY DIGIT) is de informatie eenheid in het tweetallig stelsel. Praktisch voorkomende waarden variëren zo van 30 tot 50 bits. Zo een standaard aantal noemen we dan *WOORDLENGTE*. In de regel is ook het geheugen hieraan aangepast, zodat een woord juist dit standaardaantal bits kan bevatten. Het voordeel van dit direkte binaire systeem is, dat de machine, en wel speciaal het rekenorgaan, zo eenvoudig mogelijk wordt. Een complikatie is, dat wij mensen natuurlijk in het decimale stelsel blijven werken, hetgeen betekent, dat bij invoer van gegevens en uitvoer van resultaten omzetting van talstelsels moet plaats vinden. Weliswaar kan dit ook door de machine gedaan worden, maar het kost tenslotte tijd en moeite.

Voor wetenschappelijke berekeningen, waarbij in de regel veel gerekend moet worden met relatief weinig invoergegevens, terwijl veelal ook de resultaten van beperkte omvang zijn, is dit bezwaar niet groot. Vooral in deze sektor vinden we dan ook de zuiver binair werkende machines.

Bij administratieve problemen daarentegen is de situatie in de regel precies andersom. Hier treft men grote hoeveelheden invoergegevens, relatief kleine berekeningen en vaak weer veel resultaten, die uitgevoerd moeten worden. Nemen we als voorbeeld een voorraad verschillende artikelen, waarvan per artikel gegeven zijn het aantal en de prijs per eenheid. We willen berekenen per artikel de voorraad uitgedrukt in een geldbedrag. We moeten nu als gegevens invoeren alle aantallen benevens alle eenheidsprijzen. De gehele berekening bestaat per artikel slechts uit één vermenigvuldiging. Tenslotte moeten alle geldbedragen weer worden uitgevoerd. Het is duidelijk dat in dit geval de omrekening van 10-tallig naar 2-tallig, bij invoer en omgekeerd bij uitvoer, zeer zwaar gaat wegen ten opzichte van de werkelijke gewenste berekening.

DECIMAAL-BINAIRE-CODERING

Als we convertering van talstelsel zouden kunnen vermijden, desnoods ten koste van wat inefficienter en duurder echt rekenwerk, dan zijn we toch nog voordeliger uit. Dit soort

overwegingen hebben geleid tot de construktie van de zgn. decimaal-binair werkende machines. Men heeft daarvoor een oplossing gevonden waarbij de machine technisch wel opgebouwd blijft uit 2-tallige elementen (bits), maar de decimale waarde van getallen toch weer in direkte vorm beschikbaar is. Beschouwen we daartoe eerst nog eens de binaire voorstelling van de getallen 0 t/m 9.

0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

We zien daaruit dat we alle decimale cijfers dus kunnen voorstellen met 4 bits. Met opzet hebben we hier ook voor de kleinere decimale cijfers toch 4 bits aangegeven, ook al bevatten deze "vooraan" nullen. Beschouwen we nu zo'n groepje van 4 bits, dat wel *TETRADE*genoemd wordt, als een nieuwe eenheid, dan kunnen we met deze eenheden dus wel alle decimale cijfers gebruiken.
Hebben we een getal van meerdere decimale cijfers, dan komt dit dus ook te bestaan uit meerdere tetraden.

Bijvoorbeeld het getal 865 ziet er als volgt uit :
100001100101

Voor de duidelijkheid kunnen we beter de tetraden iets uit elkaar schrijven waardoor we de afzonderlijke decimale cijfers beter herkennen.

1000	0110	0101
8	6	5

Door vergelijk geven we ook nog de zuiver binaire voorstelling van 865 n.l. : 1101100001

Machines waarbij men een groepje bits gebruikt om één decimaal cijfer voor te stellen, noemt men wel decimaal-binaire of vaak kortweg decimale machines. In het andere geval spreekt men dan van zuiver binaire of kortweg binaire machines.
Wanneer we de twee voorstellingswijzen van het getal 865 in het gegeven voorbeeld vergelijken dan blijkt dat we voor de zuiver binaire voorstelling slechts 10 bits nodig hebben tegenover 12 in de tetradenvorm. Dit komt omdat met 4 bits 16 combinaties gevormd kunnen worden. Van deze 16 mogelijkheden gebruiken we er maar 10. Bij dezelfde grenzen van nauwkeurigheid hebben we in een decimale machine dus meer bits nodig dan in een zuiver binaire.

Kijken we even naar het rekenorgaan van een dergelijke decimaal-binaire machine. We nemen als voorbeeld een een-

voudige optelling van 2 getallen ieder van 4 decimale cijfers.

```
 4863
 3939 +
 8802
```

Het voorbeeld is zo gekozen dat er bij deze optelling decimale overdrachten van de ene naar de andere cijferplaats optreden. Door dit feit kunnen we in de decimaal-binaire representatie, ondanks het feit dat per cijfer de voorstelling nog steeds zuiver binair is, toch niet zonder meer binair optellen omdat op die manier de decimale overdracht verloren gaat.
Tellen we aanvankelijk per tetrade wel binair op, dan zal juist in die gevallen waarbij eigenlijk een decimale overdracht optreedt, een combinatie van bits ontstaan die geen representatie van een decimaal cijfer vormt. Zelfs kan deze combinatie vijf bits gaan bevatten. Na de eerste aanvankelijke optelling per tetrade moeten we de zaak dus herzien voor die tetraden, die niet toegelaten zijn, door ze met binair 10 te verminderen en 1 op te tellen bij de volgende tetrade. Bij een doorlopende decimale overdracht moeten we dit spel eventueel nog een aantal keren herhalen. De optelling verloopt dan dus als volgt.

```
 0100          1000          0110          0011
 0011 +        1001 +        0011 +        1001 +
 0111          10001         1001          1100
    1 +         1010 -           1 +         1010 -
 1000           0111          1010          0010
                   1 +        1010 -
 1000          1000          0000          0010
   8             8             0             2
```

Men kan dus verwachten dat het rekenorgaan van een decimaal binaire machine ingewikkelder wordt dan van een overeenkomstige zuiver binaire machine.

Nu was bij het hier gegeven voorbeeld per decimaal cijfer, dus per tetrade, nog wel een zuiver binaire representatie aangehouden. Voor de construktie van het rekenorgaan, is dat echter nauwelijks een voordeel te noemen.
Wanneer men echter bij de construktie van een rekenmachine afgestapt is van het idee van een zuiver binaire machine en per decimaal cijfer een tetrade wil gaan gebruiken dan is er in principe geen enkel bezwaar om voor deze tetradevoorstelling een geheel andere dan de zuiver binaire te kiezen. Immers met 4 bits hebben we 16 mogelijkheden. Hiervan hebben we slechts 10 combinaties nodig, die we dan nog naar eigen willekeur mogen toekennen aan de 10 decimale cijfers. Op deze wijze hebben we 16x15x14x13x12x11x10x9x8x7= 29059430400 mogelijke combinaties. In plaats van combinaties spreekt men in de rekenmachinetechniek dan over de

toegepaste code.

Van de ruim 29 miljard mogelijke codes worden er in de praktijk slechts enkele gebruikt. De keuze berust hoofdzakelijk op argumenten van technische aard bij het ontwerpen van de machine. We willen hier volstaan met als voorbeeld één andere code te noemen en wel de code, die bekend staat als excess-three code.

EXCESS-THREE CODE
Deze code ziet er als volgt uit.

0	0011	5	1000
1	0100	6	1001
2	0101	7	1010
3	0110	8	1011
4	0111	9	1100

Deze code is dus zo gekozen dat steeds de binaire waarde van de vier bits 3 meer is dan de waarde van het bijbehorende decimale cijfer, vandaar de naam excess-three code. Een van de voordelen van de code is bijvoorbeeld dat alle cijfers van 5 of meer beginnen met 1. Dit kan van belang zijn bij afrondingskwesties. Wanneer we in deze code twee tetraden gewoon binair optellen dan ontstaat een overloop naar de vijfde binaire bitplaats juist alleen in die gevallen waarin ook bij de decimale cijfers juist een decimale overdracht ontstaat. De decimale overdracht is dus makkelijker te detecteren. Deze code is zgn. 9 COMPLEMENTAIR. D.w.z. wanneer twee decimale cijfers samen 9 zijn dan vormen na binaire optelling de twee overeenkomstige tetraden juist de combinatie 1111. Dit kan van belang zijn bij negatieve getalrepresentatie. Tot zover een bespreking van deze tetradencodes.

ENKELE ANDERE CODES
Nu willen we nog aandacht schenken aan enkele andere codes, waarbij per decimaal cijfer gebruik gemaakt wordt van meer dan 4 bits. Hiervoor zijn in principe 2 verschillende redenen aan te wijzen :

1) De meerdere mogelijkheden worden gebruikt voor kontrole doeleinden
2) De meerdere mogelijkheden worden gebruikt om ook andere symbolen dan alleen de decimale cijfers voor te stellen.

Allereerst de kontrole mogelijkheid. Wanneer we niet 4 maar 5 bits per decimaal cijfer beschikbaar stellen, dan zouden we dit als volgt kunnen doen. Eerst kiezen we een of andere 4 bits code bijvoorbeeld de reeds besproken excess-three code.
Aan iedere tetrade voegen we 1 extra bit toe zodat per groepje van 5 bits *(PENTADE)* steeds het aantal enen even is. Er ontstaat dan de volgende code.

0	00011	5	11000
1	10100	6	01001
2	00101	7	01010
3	00110	8	11011
4	10111	9	01100

Dit extra bit wordt wel genoemd het *PARITEITSBIT* (parity bit) en deze vorm van controle mogelijkheid de pariteitscontrole. Technisch bouwt men in de computer dan zodanige schakelingen in, dat steeds na ieder transport van een cijfer van één deel van de machine naar een ander deel een dergelijk pariteitsbit gecontroleerd wordt op zijn juistheid. Klopt het niet dan volgt er een alarmmelding, waardoor men geattendeerd wordt op een of andere technische fout.

In plaats van de even pariteitscontrole kent men ook de oneven pariteitscontrole waarbij het pariteitsbit dan juist zo gekozen is dat het aantal enen oneven is. In ieder geval blijft bij pariteitscontrole een of ander gekozen 4 bits code gehandhaaft en alleen 1 extra bit toegevoegd. We kunnen echter bij de keuze van de code rechtstreeks uitgaan van een pentade en uit de nu 32 ter beschikking staande mogelijkheden een zo goed mogelijke keus doen. Een voorbeeld daarvan is de zgn. 2 uit 5 code.

TWEE UIT VIJF CODE

Bij een dergelijke code zijn er van de vijf bits steeds 2 enen en 3 nullen. Deze 2 enen en 3 nullen kan men precies op 10 verschillende manieren combineren, en deze 10 combinaties dus toekennen aan de 10 decimale cijfers. Bedenk wel dit kan dan wel nog altijd op 10 (10 faculteit) verschillende manieren. Een van de mogelijke 2 uit 5 codes is de volgende.

0	00011	5	10100
1	00101	6	11000
2	00110	7	01001
3	01010	8	10001
4	01100	9	10010

Deze uitvoering noemt men wel de walking code naar aanleiding van de 2 enen die steeds beurtelings door de code "lopen" wanneer men van 0 naar 9 gaat. Deze code wordt wel gebruikt als het rekenorgaan technisch uit magneetkernen is samengesteld. Dit is overigens een technische kwestie. Let wel dat bij iedere 2 uit 5 code de controle iets beter is dan bij een pariteitscontrole.
Bij beide typen controles wordt een fout in 1 bit geconstateerd. Echter wanneer 2 nullen enen worden en omgekeerd faalt de pariteitscontrole, terwijl een 2 uit 5 code deze fout wel herkent. Al dit soort codes, die in staat zijn bepaalde fouten te herkennen, noemt men wel foutdetecterende codes *(error detecting codes)*.

Nu kan men nog verder gaan en in plaats van 5 bits nog meer bits per decimaal cijfer gaan gebruiken. Men kan dan als het ware een controle over een controle gaan uitvoeren. Het voordeel daarvan kan zijn dat bij het optreden van een enkelvoudige fout de machine technisch zelf kan constateren wat er fout is en dit zodanig zelf kan herstellen. Dat kan van belang zijn als de berekeningen koste wat het kost doorgang moeten vinden.
Natuurlijk vindt ook wel registratie plaats van iedere keer dat de machine zelf een correktie aanbrengt, maar de technische reparatie kan dan eventueel wachten op een gunstiger tijdstip terwijl de berekening onmiddelijk doorgaat. Dergelijke codes worden wel fout corrigerende codes *(error correcting codes)* genoemd. Dergelijke codes zullen we hier verder niet bespreken.

We keren nu terug naar de 2e reden waarom men soms voor een decimaal cijfer meer dan 4 bits gebruikt. Vooral voor administratieve problemen, maar tenslotte ook wel voor wetenschappelijke berekeningen, willen we ook kunnen werken met letters en bepaalde leestekens zodat ook namen, tekst, leestekens, e.d. zowel gelezen als uitgevoerd kunnen worden en in bepaalde gevallen ook verwerkt kunnen worden. We zullen ons beperken tot de 10 decimale cijfers, alle letters van het alfabet maar wel van 1 lettertype en verder enige leestekens. Dan komen we al gauw tot ongeveer 45 verschillende karakters, die we willen voorstellen. Hiervoor hebben we tenminste 6 bits *(hexade)* nodig. Dit aantal is tot op heden voor veel machines ook gebruikelijk. Een dergelijke code noemt men dan wel een ALFANUMERIEKE code, in tegenstelling tot de NUMERIEKE codes, waarbij alleen de cijfers voorgesteld kunnen worden. Ook in SERA zullen we een dergelijke alfanumerieke code in de vorm van een hexade code ontmoeten. Deze code treft U aan als aparte bijlagebij dit hoofdstuk.
Beschouwen we iedere hexade als een binair getal dan zijn de eerste 10 waarden gebruikt voor de 10 decimale cijfers, de daarop volgende 26 waarden bestemd voor 26 letters en de overige waarden vrij willekeurig toegewezen aan diverse leestekens. Dus bijvoorbeeld : 0 is 000000
9 is 001001
A is 001010 enz.

BYTE
Er bestaat op het ogenblik bij enkele nieuwe type computers een tendens om een groep van 8 bits *(oktade)* als nieuwe eenheid te hanteren. Men gebruikt hiervoor wel de naam byte. Opgevat als alfanumerieke code geeft de byte met 8 bits dus 256 combinaties. Daarmee heeft men mogelijkheden voor de 10 decimale cijfers alsmede meerdere lettertypen en een groter aantal andere tekens. Een byte kan men echter ook opvatten als een combinatie van 2 tetraden en daarmee ingeval van numeriek werk in 1 byte ook 2 decimale cijfers onderbrengen volgens een of andere numerieke tetracode. Deze methode van werken komt dus een optimaler

gebruik van de bits van het geheugen ten goede.

VARIABELE EN VASTE WOORDLENGTE

We willen thans het probleem van de geheugen indeling bespreken. In de praktijk heeft men daarvoor vaak volkomen verschillende oplossingen gekozen. In feite zijn deze oplossingen allemaal naar voren gekomen in het streven het geheugen zo effectief mogelijk te kunnen gebruiken, aangezien het geheugen nog altijd een van de kostbaarste delen van de computer vormt. Zien we even van de zojuist genoemde byte af, dan lijkt het uit gebruiksoverwegingen aantrekkelijk om het geheugen op te bouwen uit hexaden als eenheid. Men heeft dan de vrijheid om iedere eenheid naar believen te gebruiken voor een cijfer, een letter of een leesteken. Zo gauw men aan het rekenen slaat, krijgt men te maken met getallen die dan praktisch altijd uit een aantal cijfers bestaan, zodat men ook een aantal hexaden nodig heeft. Nuzal dat de ene keer meer cijfers zijn dan een andere keer. Machines met de hexade als eenheid zijn dan ook bijna altijd zo ingericht dat men het aantal benodigde hexaden van probleem tot probleem kan wijzigen en eigenlijk zelf kan kiezen. Men noemt dit type machine dan een machine met variabele woordlengte.

Bij deze machines is de geheugenadressering zo ingericht dat iedere hexade een geheugenadres heeft. Dit brengt met zich mee dat de plaats van een getal in een dergelijk geheugen nu principieel bestaat uit 2 onafhankelijke grootheden, n.l. het adres van het eerste cijfer van het getal en het adres van het laatste cijfer. Bij verschillende fabrikanten heeft men deze complikatie ook op verschillende manieren opgelost.

Een eerste methode is om in een opdracht die betrekking heeft op een bepaald getal wel op te nemen het adres van het eerste cijfer van het getal. Bedenk daarbij dat dit adres in de instruktie op zichzelf dan uit een aantal symbolen bestaat omdat een adres wel uit 4 of 5 decimale cijfers kan bestaan. In plaats van volledig ook het tweede adres aan te geven volstaat men met aan te geven uit hoeveel cijfers het getal bestaat.

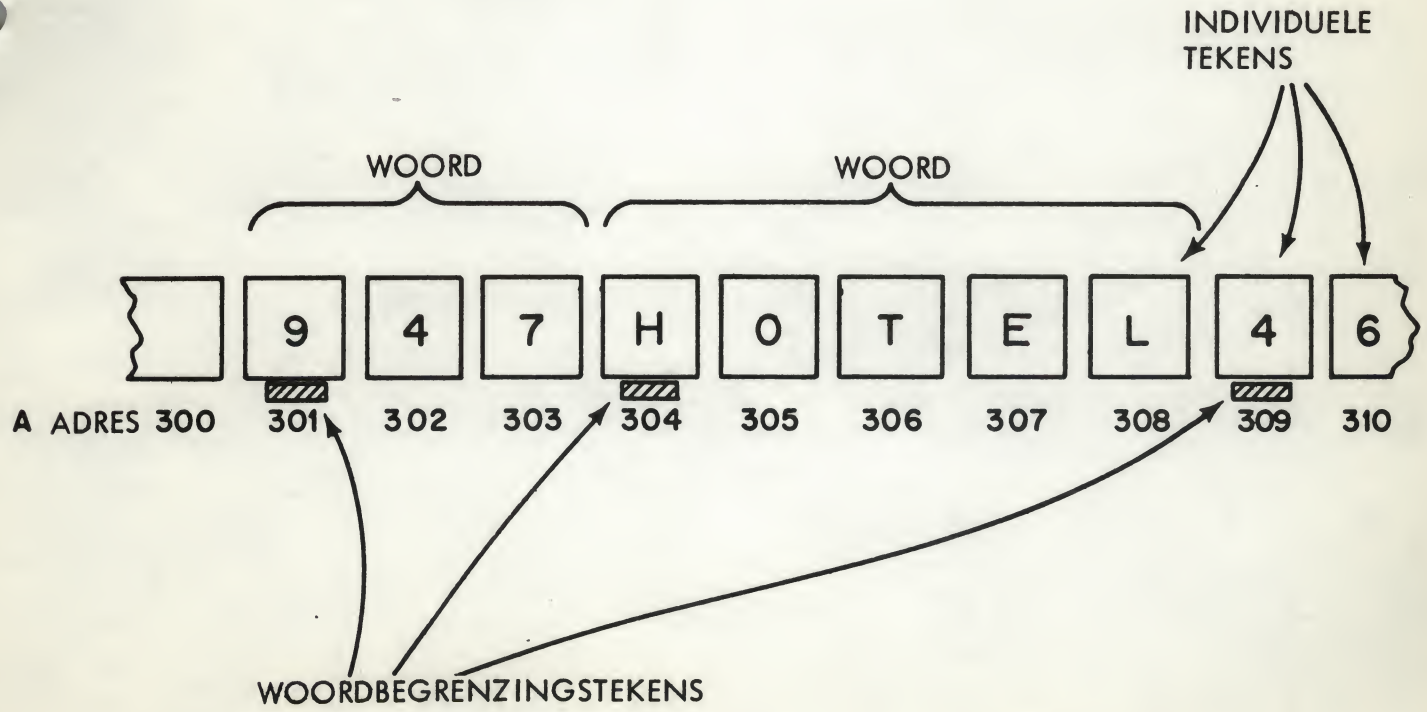
Dit doet men dan b.v. in een binaire code en kan dit dan in 1 hexade opbergen. Dit noemt men wel de tellercode.

Een andere methode is het werken met zgn. grenssymbolen. Wanneer we achter elkaar in het geheugen aan aantal getallen plaatsen, dan worden deze niet aansluitend gezet, maar steeds wordt één symbool, dat dan geen cijfer voorstelt, tussen twee opvolgende getallen geplaatst. Ieder getal staat dan als het ware opgesloten tussen twee begrenzingen, de grenssymbolen. Als grenssymbolen kan men dan bijvoorbeeld gebruiken het + of - teken voor ieder getal. Op die manier vormt het teken van het volgend getal dan de afsluiting van het vorige. Zijn alle getallen uit de aard van het probleem toch positief dan zou men geneigd zijn ze weg te laten. Echter als grenssymbool moeten ze nu toch gehandhaaft worden, hetgeen op zichzelf een zekere verkwisting van geheugenruimte betekent. Daar staat tegenover dat men in een instruktie nu kan volstaan met alleen het beginadres van het gewenste getal aan te geven.

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0000	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	0010	0011	0012	0013	0014	0015
0010	0016	0017	0018	0019	0020	0021	0022	0023	0024	0025	0026	0027	0028	0029	0030	0031
0020	0032	0033	0034	0035	0036	0037	0038	0039	0040	0041	0042	0043	0044	0045	0046	0047
0030	0048	0049	0050	0051	0052	0053	0054	0055	0056	0057	0058	0059	0060	0061	0062	0063
0040	0064	0065	0066	0067	0068	0069	0070	0071	0072	0073	0074	0075	0076	0077	0078	0079
0050	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	0090	0091	0092	0093	0094	0095
0060	0096	0097	0098	0099	0100	0101	0102	0103	0104	0105	0106	0107	0108	0109	0110	0111
0070	0112	0113	0114	0115	0116	0117	0118	0119	0120	0121	0122	0123	0124	0125	0126	0127
0080	0128	0129	0130	0131	0132	0133	0134	0135	0136	0137	0138	0139	0140	0141	0142	0143
0090	0144	0145	0146	0147	0148	0149	0150	0151	0152	0153	0154	0155	0156	0157	0158	0159
00A0	0160	0161	0162	0163	0164	0165	0166	0167	0168	0169	0170	0171	0172	0173	0174	0175
00B0	0176	0177	0178	0179	0180	0181	0182	0183	0184	0185	0186	0187	0188	0189	0190	0191
00C0	0192	0193	0194	0195	0196	0197	0198	0199	0200	0201	0202	0203	0204	0205	0206	0207
00D0	0208	0209	0210	0211	0212	0213	0214	0215	0216	0217	0218	0219	0220	0221	0222	0223
00E0	0224	0225	0226	0227	0228	0229	0230	0231	0232	0233	0234	0235	0236	0237	0238	0239
00F0	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	0250	0251	0252	0253	0254	0255

CONVERSIETABEL VAN HEXADECIMAAL NAAR DECIMAAL

WERKGEHEUGEN VAN EEN COMPUTER MET VARIABELE WOORDLENGTE



VLAGBIT OF WORDMARK

Een methode die enigszins op het bovenstaande lijkt is het werken met een zgn. vlagbit of wordmark. Daarbij wordt in het gehele geheugen consequent iedere hexade voorzien van een extra bit. In feite gebruikt men dan dus 7 in plaats van 6 bits per karakter. Wenst men voor een bepaald probleem het geheugen nu in te delen voor getallen van een bepaalde lengte dan wordt voor ieder getal het vlagbit van het laatste cijfer 1 gemaakt en voor de overige cijfers 0. In de instruktie kan men dan weer volstaan met alleen het adres van het eerste cijfer. Juist omdat een machine met variabele woordlengte zo geconstrueerd moet worden dat iedere keer met andere lengten van getallen gewerkt moet kunnen worden is een dergelijke machine praktisch altijd uitgevoerd in de vorm van een seriemachine.

SERIEMACHINE

Hieronder verstaat men een machine, die bij een rekenbewerking de verschillende cijfers in de tijd achter elkaar verwerkt. Het voordeel daarvan is dat het rekenorgaan dan betrekkelijk eenvoudig kan zijn. Immers voor bijvoorbeeld een optelling kan men volstaan met één optelschakeling voor slechts een cijfer. Moeten we dan 2 getallen, ieder bestaande uit meerdere cijfers, optellen, dan kan steeds dezelfde schakeling achtereenvolgens voor ieder cijfer gebruikt worden.
Door de eenvoud is daardoor de seriemachine relatief goedkoop. Als nadeel geldt dat een seriemachine ook relatief langzaam werkt. Is voor de problemen die op een machine verwerkt moeten worden, de rekensnelheid van wezenlijk belang, dan is een seriemachine minder aantrekkelijk. Immers willen we voor grotere getallen van bijvoorbeeld 10 of meer cijfers toch een grote rekensnelheid behalen dan moeten we bij een seriemachine de snelheid per verwerkt cijfer extreem snel gaan maken hetgeen dan ten koste gaat van bijzonder ingewikkelde en tevens dure schakelingen. In zo'n geval geven we liever de voorkeur aan een parallelmachine.

PARALLELMACHINE

Een dergelijke machine is gebaseerd op het gelijktijdig verwerken van een groter, maar wel is waar vast aantal cijfers. Van machine tot machine kan dit verschillen, maar ligt in de regel in de orde van 10 tot 15 decimale cijfers of ingeval van een zuiver binaire machine in de orde van 40 tot 60 bits. Door zijn construktie is een parallelmachine daardoor in de eigenschappen ook juist tegengesteld aan een seriemachine. Door het parallel verwerken van alle cijfers moet iedere cijferplaats nu ook een afzonderlijke schakeling bezitten.
Men kan dus bij een parallel machine meer onderdelen verwachten dus ook een duurder machine. Als voordeel geldt natuurlijk dat juist door de parallel werking men ook een grotere verwerkingssnelheid mag verwachten.

Juist omdat een parallel machine gebaseerd is op de gelijktijdige verwerking van een vast aantal cijfers is een parallel machine in de regel ook uitgevoerd als een vaste woordlengte machine.

Men dient echter in het oog te houden dat parallel en vaste woordlengte elkaar niet behoeven te dekken. Er bestaan enkele vaste woordlengte machines die uit economische overwegingen als serie machine zijn uitgevoerd. Omgekeerd is het ook denkbaar een variabele woordlengte machine als parallel machine uit te voeren. De eerder genoemde combinaties zijn echter de meest gebruikelijke. Bij een vaste woordlengte machine, afgezien of het nu serie of parallel is, wordt iedere bewerking steeds uitgevoerd op een vast aantal cijfers. Ook het geheugen is dan aangepast aan dit vaste aantal, waarbij dit aantal dan gehanteerd wordt als eenheid. Deze eenheden worden weer genummerd en vormen dan op deze wijze de adressering. Het verschil met variabele woordlengte is echter dat we nu zonder meer met één adres de plaats van een compleet getal kunnen aangeven.

Vaste woordlengte betekent niet dat men niet zou kunnen werken met een kleiner aantal cijfers dan het vaste aantal. Alleen worden in dat geval de eerste cijfers dan 0. Bijvoorbeeld een machine met een vaste woordlengte van 9 decimale cijfers zal het getal 78654 verwerken als 000078654.
Werkt men dus veel met kleine getallen, althans kleiner dan het vaste aantal van de betrokken machine, dan wordt dus ook een gedeelte van de geheugenruimte nodeloos in beslag genomen. Echter voor het zuivere rekenwerk is een vaste woordlengte machine toch wel eenvoudiger te hanteren en door parallel transport per gehele eenheid in de regel ook sneller te maken.
Wat de woordindeling van een dergelijke machine betreft, is deze in de eerste plaats gebaseerd op het gewenste aantal cijfers, hetzij binair hetzij decimaal in tetraden vorm. Denken we in de eerste plaats aan een decimale machine, dan kunnen we als woordlengte dus bij voorkeur een veelvoud van 4 bits verwachten. Anderzijds zal het bij een vaste woordlengte machine ook wel nodig zijn om met alfanumerieke symbolen te manipuleren. Dit laatste geeft dan een voorkeur voor een veelvoud van 6 bits als woordlengte. Daaruit volgt dat voor decimale machines een veelvoud van 12 bits het meest aantrekkelijk is. Veel voorkomende waarden zijn dan ook 24, 36 of 48 bits. Om misverstand te voorkomen, er komen ook wel andere woordlengten voor maar uit een oogpunt van efficiënt geheugengebruik zijn deze aantallen aantrekkelijk.

VOORSTELLING VAN NEGATIEVE GETALLEN

Tot slot van de bepreking over getallenrepresentatie in een elektronische rekenmachine moeten we nog enige aandacht schenken aan de voorstellingswijze van negatieve getallen. De gebruikelijke voorstelling in ons decimale stelsel is

daarbij het zgn. *modulus-systeem*. Overeenkomstige positieve en negatieve getallen worden daarbij voorgesteld door gelijke cijfercombinaties. Het onderscheid wordt gevormd door 2 aparte symbolen n.l. het + en het - teken. Er zijn ook andere systemen denkbaar waarvan we er hier 2 willen bespreken, omdat de overeenkomstige systemen in het binaire stelsel juist de meeste toepassing in rekenmachines vinden. Beschouwen we alle getallen, die gevormd kunnen worden door de combinaite van 4 cijfers. In normaal gebruik stellen we daar dan mee voor alle getallen van 0 t/m 9999. Deze 10000 combinaties van cijfers zouden we echter ook kunnen gebruiken om daarmede aan te geven de reeks van getallen van -5000 t/m +4999. Van alle mogelijke combinaties nemen we daarbij één speciale. De getallen +0 t/m +4999 worden voorgesteld door 0000 t/m 4999, de negatieve getallen -5000 t/m -1 door 5000 t/m 9999. De nieuwe voorstellingswijze van een negatief getal kunnen we daarbij bepalen door van de modulus van het negatieve getal het complement te nemen ten opzichte van 10000. Omdat we hier het complement nemen ten opzichte van een 10-macht (in dit geval 10^4 omdat we met 4 cijfers werken) spreken we over 10-complementsysteem. Enkele voorbeelden :

+ 365	wordt	0365
+ 3784	wordt	3784
- 75	wordt	9925
- 763	wordt	9237
- 2836	wordt	7164
- 4157	wordt	5843

Het "teken" van het getal kunnen we nog steeds herkennen aan het eerste cijfer. Is dit kleiner dan 5 dan is het getal negatief en is het eerste getal groter of gelijk aan 5 dan stelt de gehele combinatie van cijfers een negatief getal voor. Het voordeel van dit systeem is dat met dit "tekencijfer" op dezelfde manier gerekend kan worden als met de overige. Nemen we als voorbeeld enkele optellingen en aftrekkingen en vergelijken die in beide systemen.

+ 365		0365		-1987		8013	
+2987	+ wordt	2987	+	-2753	- wordt	7265	+
+3352		3352		-4722		5278	

In dit laatste geval ontstaat eigenlijk een 1 als 5e cijfer. Omdat we echter slechts met 4 cijfers werken laten we deze zonder meer weg en verkrijgen dan automatisch het juiste antwoord.
Nog sprekender is het geval wanneer beide cijfers verschillende tekens hebben. In het modulussyteem wordt optellen nu eigenlijk aftrekken, in het 10-complementsysteem blijft optellen optellen.

+ 4768		4768		+ 2658		2658	
- 3537	+ wordt	6463	+	- 4935	+ wordt	5065	+
+ 1231		1231		- 2277		7723	

Ook aftrekken blijft in dit systeem aftrekken.

+ 3768		3768		- 2163		7837	
+ 4593	- wordt	4593	-	- 3829	- wordt	6171	-
- 825		9175		+ 1666		1666	

Een nadeel van dit systeem is, dat de negatieve representatie van een getal alleen verkregen kan worden door een complete aftrekking van 0000.

		0000				0000	
		3784	-			6212	-
-3784	=	6216		+3784	=	3784	

Een iets ander systeem komt aan deze moeilijkheid tegemoet. Dit is het *9-complementsysteem*. Hier bepaalt men de negatieve representatie van een getal door, voor ieder cijfer afzonderlijk, het complement ten opzichte van 9 te nemen.
Voorbeeld :

+ 3584	wordt	3584
- 3584	wordt	6415

In dit systeem doet zich nu de merkwaardigheid voor, dat ons getal 0 tweemaal voorkomt n.l. als 0000 en als 9999. Dit noemen we dan wel +0 en -0, hoewel de numerieke waarde natuurlijk nul is. Een vervelende consequentie hiervan is bovendien, dat bij optellingen en aftrekkingen, waarbij 0 gepasseerd wordt, in feite +0 en -0 gepasseerd worden en men daardoor in het antwoord 1 te kort komt. Dit gebeurt telkens wanneer men de capaciteit van het aantal mogelijke cijfers (in onze voorbeelden 4) overschrijdt.
Voorbeeld :

- 3584		6415	
+ 4735	+ wordt	4735	+
+ 1151		11150	
		└--→1	
		1151	

Men lost deze moeilijkheid op door de overdracht die geheel links ontstaat rechts weer bij te voegen. Men noemt dit *rondlopende overdracht* (end-around carry)

Nog enkele voorbeelden :

- 2584		7415	
- 1732	+ wordt	8267	+
- 4316		15682	
		└--→1	
		5683	

Ir. D.H. Wolbers

```
+ 2584          2574
+ 1732  +  wordt 1732  +
+ 4316          4316
```

In dit laatste geval treedt geen overdracht, op de meest linkse plaats, op en wordt dus ook niet gecorrigeerd. Ook bij aftrekken moet een eventueel "leen" rondgekoppeld worden.

```
- 3584          6415
- 1732  -  wordt 8267  -
- 1852          8148
                   1
               ------
               8147

- 1732          8267
- 3584  -  wordt 6415  -
+ 1852          1852
```

OVERZICHT

Tot slot een overzicht van enkele getalwaarden in de 3 stelsels onder de aanname, dat we met getallen van 4 cijfers werken.

MODULUSSYSTEEM	10-COMPLEMENTSYSTEEM	9-COMPLEMENTSYSTEEM
+ 4999	4999	4999
+ 3000	3000	3000
+ 1	0001	0001
+ 0	0000	0000
- 0	bestaat niet	9999
- 1	9999	9998
- 3000	7000	6999
- 4999	5001	5000
- 5000	5000	bestaat niet

Het belang van deze nieuwe vormen van getalrepresentatie geldt in het bijzonder, wanneer hun equivalenten in het tweetallig stelsel beschouwd worden. Ook hier kennen we het modulussysteem. Met het 10-complementsysteem komt dan overeen het 2-complementsysteem, terwijl het 9-complement zijn partner vindt in het 1-complement. In het modulussysteem kunnen we eenvoudig aan de bits die het getal voorstellen, een extra bit toevoegen waarvan 0 of 1 dan overeenkomen met x + of -. Algemeen gebruikelijk is daarbij dat 0 overeenkomt met +. In dit modulussysteem verandert dus alleen dit "tekenbit" als we een getal van teken veranderen, terwijl de "cijferbits" ongewijzigd blijven.
In het 2-complementsysteem vinden we de negatieve representatie door het gegeven getal van 0 af te trekken en

daarbij een eventuele "leen" geheel links buiten beschouwing te laten, zoals ook in het 10-complementsysteem.

DECIMAAL SYSTEEM	BINAIR MODULUS SYSTEEM	2-COMPLEMENT SYSTEEM	1-COMPLEMENT SYSTEEM
+ 7	0111	0111	0111
+ 6	0110	0110	0110
+ 5	0101	0101	0101
+ 4	0100	0100	0100
+ 3	0011	0011	0011
+ 2	0010	0010	0010
+ 1	0001	0001	0001
+ 0	0000	0000	0000
- 0	1000	bestaat niet	1111
- 1	1001	1111	1110
- 2	1010	1110	1101
- 3	1011	1101	1100
- 4	1100	1100	1011
- 5	1101	1011	1010
- 6	1110	1010	1001
- 7	1111	1001	1000
- 8	bestaat niet	1000	bestaat niet

In het 1 complementsysteem verkrijgen we de negatieve waarde van een getal door eenvoudig alle nullen in enen en alle enen in nullen te veranderen. Men noemt dit systeem daarom ook wel het inverse systeem. Deze omzetting is technisch bovendien zeer eenvoudig uit te voeren. Werkt men in een decimale machine en heeft men daarbij een code gekozen, die 9-complementair is zoals de eerder als voorbeeld gegeven excess three code dan kan het 9-complement ook daar technisch gerealiseerd worden door inverteren.
In de meeste machines geeft de rondlopende overdracht technisch ook geen moeilijkheden. Dit systeem wordt in zuiver binair werkende machines dan ook veel toegepast. Het nadeel van de +0 en -0 blijft echter en dit is een eigenschap, die ook bij het programmeren soms vervelende consequenties heeft. In bovenstaande tabel zijn deze drie systemen met elkaar vergeleken voor het geval van 4 bits inclusief het "tekenbit". We merken daarbij op dat in alle drie systemen steeds het eerste bit fungeert als tekenbit met 0 als + en 1 als -.
In het modulussysteem moet dit tekenbit bij rekenbewerkingen echter geheel gescheiden behandeld worden.
Zowel in het 1- als 2-complementsysteem is de behandeling van het tekenbit echter volkomen gelijk aan de overige. Dit geeft althans technisch grote voordelen. In de tabel voor het 1-complementsysteem is nu ook te zien dat een 1 als tekenbit niet altijd een garantie is voor echt negatief zijn van een getal, n.l. juist in het geval van -0. Wil men in dit systeem dus nagaan of een getal kleiner is dan 0 dan moet bovendien worden nagegaan of het niet toevallig -0 is.

WOORDSTRUCTUUR VAN DE SERA

Met de kennis van het voorafgaande kunnen we nu de woordstructuur van de SERA beschrijven. De SERA werkt als een binair-decimaal machine maar kan ook alfanumerieke informatie verwerken. Wat betreft de alfanumerieke code wordt gewerkt met karakters van 6 bits. Per karakter kunnen dus 2^6 is 64 verschillende symbolen worden voorgesteld. Dit zijn in de eerste plaats de 10 decimale cijfers en de 26 letters van het alfabet. Verder hebben een aantal andere tekens een binaire combinatie toegewezen gekregen. Een volledige vertaaltabel ontvangt U als afzonderlijke bijlage bij deze kursus.
De woordlengte van de SERA is zodanig, dat een woord 8 alfanumerieke karakters kan bevatten. In totaal bevat een woord dus 48 bits, echter in de vorm van 8 keer 6 bits. Dit betekent dat in deze vorm getallen van max. 8 decimale cijfers per woord gebruikt kunnen worden. Moet het getal een teken hebben dan kost dit teken reeds een volledig karakter en kunnen er nog slechts getallen van max. 7 cijfers per woord worden voorgesteld. Voor het rekenwerk betekent dit dus eigenlijk een zeer inefficient gebruik van de totale beschikbare 48 bits.
Om deze reden kent de SERA nog een tweede verdeling van een woord, die gebaseerd is op een numerieke code. In dat geval worden de 48 bits beschouwd als 12 tetraden. Welke voorstellingswijze op een gegeven moment wordt bedoeld volgt zonder meer uit het gebruik en geeft geen aanleiding tot fouten. Bij gebruik van de numerieke code heeft de meest linkse tetrade de speciale betekenis van + of - teken. Daarbij staat de combinatie 0000 voor + en 1111 voor -.
Andere combinaties hebben voor dit karakter geen betekenis. In de overige 11 tetraden worden decimale cijfers voorgesteld in normale binaire representatie.

Voorbeeld : +35791246805 wordt

0000	0011	0101	0111	1001	0001	0010	0100	0110	1000
+	3	5	7	9	1	2	4	6	8

0000	0101
0	5

Een getal dat uit minder dan elf cijfers bestaat bevat nullen op de linkse plaatsen.

Voorbeeld : + 365

0000	0000	0000	0000	0000	0000	0000	0000	0000	0011	0110	0101
+	0	0	0	0	0	0	0	0	3	6	5

SERA WERKT ALS DECIMALE MACHINE

De SERA werkt als een decimale machine, hetgeen betekent dat waar nodig decimale overdrachten van de ene tetrade naar de volgende automatisch worden verwerkt. Ook hier geldt weer, dat alleen bij zeer speciale toepassingen de

werkelijke binaire voorstelling van betekenis is. Volledigheidshalve vermelden we hier ook de voorstellingswijze van negatieve getallen. Deze werkt volgens het binaire inverse systeem, waarbij alle 48 bits ongeacht hun samenstelling geïnverteerd worden. Daardoor gaan op de tekenplaats 4 nullen over in 4 enen of 4 enen in 4 nullen. Echter ook alle andere bits worden omgedraaid. Als voorbeeld geven we nog dezelfde getallen als hierboven, in negatieve voorstelling.

Voorbeeld : -35791246805 wordt

1111 1100 1010 1000 0110 1110 1101 1011 1001 0111 1111 1010

- 3 5 7 9 1 2 4 6 8 0 5

en - 365 wordt

1111 1111 1111 1111 1111 1111 1111 1111 1111 1100 1001 1010

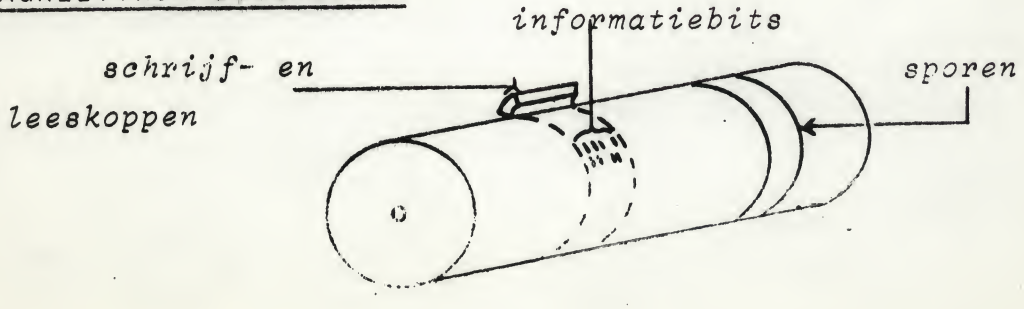
- 0 0 0 0 0 0 0 0 3 6 5

Zou men door zeer geraffineerde programmatische truc's één enkele cijfertetrade uit een woord lichten, dan moet men zich dus van deze voorstellingswijze bewust zijn. In alle andere gevallen gaat alles automatisch goed en is het volkomen onverschillig in welk systeem de machine technisch werkt.
Bij verdere besprekeing van de SERA wordt in voorkomende gevallen de binaire voorstelling dan ook niet meer aangegeven maar gewerkt volgens het gewone decimale stelsel.

3. HET GEHEUGEN

De SERA heeft een geheugencapaciteit van 10.000 woorden. De adressen zijn daarbij genummerd van 0 t/m 9999. Technisch is het uitgevoerd in de vorm van *MAGNEETRINGENGEHEUGEN*.
Ten aanzien van dit laatste kunnen we het volgende opmerken. Bij praktisch alle rekenmachines, die op het ogenblik gebruikt worden, worden slechts enkele verschillende technische principes toegepast. Hoewel de praktische uitvoeringen nog wel grote verschillen kunnen vertonen, zijn programmatisch vooral de principes van belang. We geven daarom eerst deze principes.

HET MAGNEETTROMMELGEHEUGEN

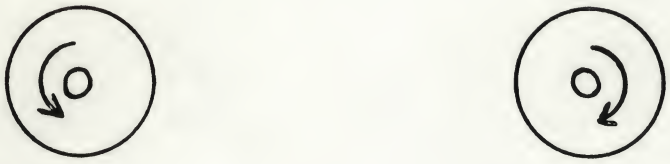


Een dunwandige metalen cilinder draait met een constant hoog toerental, bijvoorbeeld 6000 omw./min. De opppervlakte van de mantel is belegd met een dun laagje magnetisch materiaal. De totale oppervlakte wordt verdeelt in een aantal banden, die we *SPOREN OF TRACKS* noemen. Tegenover ieder spoor bevindt zich, op zeer korte afstand van de trommel, een kleine elektromagneet. Sturen we door het spoeltje hiervan een stroom, dan wordt daardoor plaatselijk een plekje van het betrokken spoor gemagnetiseerd. Is de trommel precies één omwenteling verder gedraaid dan passeert dit nu gemagnetiseerde plekje opnieuw de elektromagneet, waar dan een elektrische spanning wordt opgewekt. Dit kan verder plaatsvinden tijdens het passeren bij iedere omwenteling. Het laatste noemen we het *lezen* van de informatie, de eerste handeling het *schrijven*. De elektromagneet die dit bewerkstelligt noemen we lees- en schrijfkop. Ieder plekje, dat op deze manier gemagnetiseerd kan worden, kan dus 1 bit informatie bevatten. Per spoor kunnen we een groot, maar weliswaar beperkt aantal plekjes magnetiseren. Dit kunnen bijvoorbeeld 1200 plekjes per spoor zijn. Bij een woordlengte zoals bij de SERA betekent dit 25 woorden per spoor. Komen op de trommel 300 sporen voor, dan zou in dat geval de totale capaciteit van zo'n geheugen 7500 woorden bedragen.
Een trommelgeheugen is matig snel. In het gegeven voorbeeld duurt 1 omwenteling van de trommel 10 msec. Met 25 woorden per spoor, en dus per omwenteling, duurt het lezen of schrijven van een woord dan 1/25 x 10 msec. = 0,4 msec. = 400 msec.
Tegenover rekentijden van enkele mikroseconden bij moderne machines is dit relatief langzaam. Daarbij komt nog een groot nadeel van het trommelgeheugen. Bij de berekening van 400 msec. zijn we stilzwijgend er van uitgegaan, dat de plaats op de trommel, die we nodig hadden, zich juist tegenover de betrokken schrijf- en leeskop bevond. In het algemeen is dat niet het geval, en dan moeten we eerst wachten tot de juiste plaats voorkomt. Deze wachttijd kan in het voorbeeld max. 10 msec. zijn en bedraagt gemiddeld 5 msec., wanneer althans de informatie, die we nodig hebben kriskras op de trommel door elkaar staat. Bij het programmeren voor een machine met een dergelijk geheugen kan men daar dan gedeeltelijk rekening mee houden. Dat noemt men wel optimaal programmeren. Hier is dus een voorbeeld, waarbij de techniek van de machine invloed heeft op het programmeren. Ondanks dit nadeel worden trommelgeheugens toch nog wel gebruikt, aangezien het trommelgeheugen betrekkelijk goedkoop is.

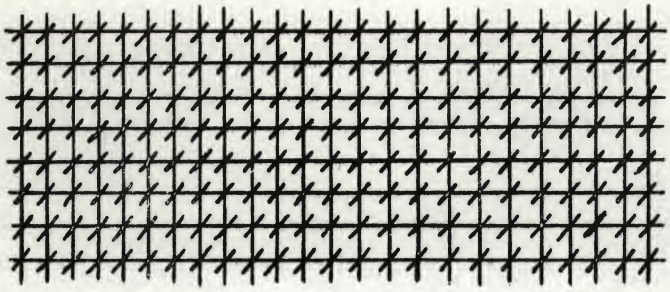
MAGNEETRINGENGEHEUGEN

De tweede vorm van geheugen is het magneetringengeheugen of ook wel kerntjesgeheugen genoemd. Dit is heden ten dage praktisch de duurste vorm van geheugen. Het bestaat uit zeer kleine ringetjes gemaakt van ferriet, een materi-

aal met bijzonder magnetische eigenschappen.



Ieder ringetje kan dan in de ene of andere richting gemagnetiseerd worden, hetgeen dan overeenkomt met een 0 of een 1. Dit heeft men in de hand door elektrische stroompjes te sturen door draden die door deze ringetjes zijn gestoken. Door ieder ringetje gaan 2 elkaar loodrecht kruisende draden en de ringetjes worden op die manier tot matten verenigd.



Een voorkomend aantal is bijvoorbeeld 64 verticale en horizontale draden, zodat per mat $64^2 = 4096$ ringetjes voorkomen. Plaatsen we dan 48 matten achter elkaar dan bereiken we op deze manier een capaciteit van 4096 woorden, zoals ook in de SERA voorkomen. De afmetingen blijven klein vanwege de kleine ringetjes, die zelf een doorsnede hebben in de orde van 1 mm. Een geheel geheugenblok van 4096 woorden heeft dan ongeveer de afmetingen van 20 x 20 x 30 cm.
Het magneetringengeheugen werkt zeer snel. Men moet daarbij rekenen met enkele microseconden. Verder zijn alle ringetjes gelijkwaardig en kan men via een elektronisch keuzesysteem alle ringetjes even vlug bereiken. Er is dus ook geen wachttijdprobleem zoals bij de trommel. Om deze redenen worden magneetringengeheugens, ondanks de hoge kostenfaktor, bij moderne machines veelvuldig toegepast. Enkele jaren geleden ging men daarbij niet verder dan 20000 à 30000 woorden. De laatste tijd is er neiging ook grotere kerngeheugens toe te passen tot enkele honderdduizenden woorden. Dit geeft dan wel enkele grotere mogelijkheden om met de machines te kunnen werken, maar zeker voor administratieve problemen en in vele gevallen ook

voor wetenschappelijke berekeningen is een dergelijk aantal totaal onvoldoende. Men beperkt dan toch het magneetringengeheugen tot bovengenoemde aantallen, maar gebruikt daarnaast, weliswaar langzamer, maar tegelijkertijd veel grotere en goedkopere vormen van geheugen. We spreken dan over snel en langzaam geheugen. Het snelle geheugen werkt direkt samen met de andere snelle elektronische onderdelen van de rekenmachine, zoals rekenorgaan en besturing. De langzame vorm van geheugen werkt op de achtergrond als een soort buffergeheugen. Het wordt alleen gebruikt om van tijd tot tijd het snelle geheugen weer bij te vullen met nieuwe informatie, of om hieruit berekende resultaten op te nemen, zodat die plaatsen weer vrij komen voor andere berekeningen. Een belangrijke vorm van langzaam geheugen is de magneetband.

DE MAGNEETBAND

Magneetbandgeheugens zijn vergelijkbaar met de bekende tape-recorders, alleen worden hier kwalitatief veel hogere eisen gesteld. Men werkt ook altijd met eniszins andere afmetingen van tape n.l. meestal ½ tape met lengten van 750m. tot 100m. Ondanks de hoge bandsnelheid van ongeveer 2.5 m./sec. kunnen bij deze grote lengten tape dan toch wachttijden optreden in de orde van minuten.
Het gebruik van magneetbanden wordt behandeld aan het einde van de kursus, waarbij ook meer gegevens worden verstrekt. In de hoofdstukken daaraan voorafgaande zullen voorbeelden worden behandeld, waarbij de SERA alleen gebruik maakt van de magneetringengeheugens.
De zeer lange wachttijden, die bij magneetbanden optreden, zijn voor verschillende fabrikanten aanleiding geweest, te zoeken naar technische oplossingen om deze tijden kleiner te maken. Alle oplossingen vertonen het gemeenschappelijke karakter, dat men de band als het ware in stukken knipt. Heeft men een bepaalde informatie nodig, dan kiest men eerst op mechanische wijze het betrokken stuk en zoekt hierop de plaats waar de gewenste informatie staat of geschreven moet worden. Voorbeelden hiervan zijn het schijvengeheugen.

SCHIJVENGEHEUGEN

Het schijvengeheugen bestaat uit een aantal ronde schijven van magnetisch materiaal en gemonteerd op een gemeenschappelijke as. Op iedere schijf wordt aan beide zijden informatie geschreven in concentrische cirkels. Met behulp van een of meer armen, waarop een schrijf-leeskop gemonteerd is, kiest men mechanisch eerst de betrokken schijf, dan op die schijf de juiste ring; daarna kan men dan lezen of schrijven.

MAGNEETKAARTGEHEUGEN

Bij het magneetkaartengeheugen is de informatie ondergebracht op kaarten van buigzaam materiaal. Op mechanische manier wordt eerst de juiste kaart gekozen, deze wordt

daarna om een snel draaiende trommel gelegd, waarna de trommel met kaart werkt als een trommelgeheugen. Na gebruik wordt de kaart weer van de trommel gelicht en in de voorraad kaarten teruggebracht.
Bij deze uitvoeringen bereikt men toegangstijden van de orde van 0,1 tot 1 sec. Deze tijden liggen dus veel gunstiger dan bij magneetbanden. Vooral de laatste jaren is daardoor het gebruik van vooral schijvengeheugens sterk toegenomen t.o.v. magneetbanden.

Men duidt schijven- en magneetkaarten geheugens vaak wel aan met de naam *RANDOM-ACCESSGEHEUGENS*. Deze benaming heeft dan betrekking op het feit dat iedere geheugenplaats gemiddeld even snel bereikbaar is. In dat verband wordt magneetbandgeheugen wel aangeduid als *SERIAL* geheugen omdat men hier principieel alle informatie achter elkaar leest of schrijft.

Tenslotte een opmerking, die geldig is voor iedere vorm van geheugenuitvoering. Bij het schrijven van nieuwe informatie ergens in het geheugen gaat automatisch de oude informatie, die op die plaats stond, verloren. Bij het lezen van informatie vanaf een plaats, blijft de informatie, die er staat onveranderd. Op iedere plaats in het geheugen staat altijd iets. Wanneer men dus de berekening van een nieuw probleem op de machine start zijn aanvankelijk de geheugenplaatsen nog gevuld met informatie van het vorige probleem waaraan de machine heeft gerekend. Met dit feit moet men ook rekening houden bij het maken van het programma.

4. HET REKENORGAAN VAN DE SERA

Het rekenorgaan van de SERA bevat 2 registers, die ieder een geheel woord kunnen bevatten. Zo'n register wordt *ACCUMULATOR* genoemd. Zij worden symbolisch aangegeven met de letters A en B. Zoals in het begin van dit hoofdstuk vermeld kan het besturingsorgaan zorgen voor de uitvoering van opdrachten, die in het geheugen staan. Een deel van deze opdrachten heeft betrekking op werkzaamheden van het rekenorgaan. De eenvoudigste daarvan zijn de haal- en brengopdrachten. Bij een haalopdracht wordt een getal uit het geheugen naar A of B gebracht. Er bestaan daartoe 2 verschillende haalopdrachten. Bij een haalopdracht gaat de oude inhoud van de betrokken accumulator verloren. De andere blijft onveranderd. De 2 brengopdrachten bewerkstelligen precies het omgekeerde. Dan wordt het getal uit A of B naar een plaats in het geheugen gebracht. De volgende soorten opdrachten zijn rekenopdrachten. De eenvoudigste hiervan zijn optellen en aftrekken.
Van ieder zijn er weer twee, n.l. 1 voor A en 1 voor B. Bij een optelopdracht in A wordt een getal uit het geheugen gehaald en opgeteld bij het getal, dat reeds in A staat. Na de optelling verschijnt het resultaat in A en is de oude inhoud van A verloren. Tegelijkertijd blijft de B-accumulator weer onaangetast. Dezelfde gang van

zaken geldt voor aftrekken in A, alsmede voor optellen in B. Willen we dus 2 getallen, die in het geheugen staan op 2 verschillende adressen, optellen met behulp van de B-accumulator, terwijl het resultaat terecht moet komen op een derde plaats in het geheugen, dan kost dit 3 opdrachten. Deze kunnen we als volgt omschrijven :
1. breng het eerste getal naar B
2. tel het tweede getal op in B
3. breng het getal van B uit naar de gewenste plaats.
Tot zover zijn A en B dus gelijkwaardig en kunnen onafhankelijk van elkaar worden gebruikt. Aangezien de genoemde opdrachten A en B nooit gelijktijdig gebruikt worden, zou men hebben kunnen volstaan met één accumulator. De reden voor 2 accumulatoren ligt bij de vermenigvuldigen deelopdracht.

VERMENIGVULDIGEN

Wanneer we een getal van 2 cijfers vermenigvuldigen met een getal van 2 cijfers, dan ontstaat in het algemeen een resultaat van 4 cijfers. Alleen wanneer beide getallen toevallig klein zijn kan een resultaat van 3 cijfers ontstaan, bijvoorbeeld 10 x 10 = 100. Bij 32 x 32 = 1024, zodat als resultaat 4 cijfers ontstaan. In de SERA kunnen we werken met getallen van 11 cijfers. Dit betekent dat na vermenigvuldiging een resultaat kan ontstaan van 22 cijfers. Men zegt dan wel : " het produkt van 2 enkele-lengte getallen is een dubbel-lengte getal ". Met " enkele-lengte " bedoelt men de woordlengte van de betrokken machine; in dit geval 11 cijfers. Voor dit doel zijn beide accumulatoren A en B aangebracht.

HOE VERLOOPT NU ZO'N VERMENIGVULDIGOPDRACHT ?

Aangenomen wordt, dat in B reeds een getal staat. De vermenigvuldigopdracht zorgt er dan voor, dat dit getal in B vermenigvuldigd wordt met een getal uit het geheugen, waarna het resultaat wordt geplaatst in A en B samen. Daarbij verschijnen de rechtse 11 cijfers van het produkt in B en de linker 11 cijfers van het produkt in A. Nu vormen A en B samen dus een getal van dubbele lengte.
Voor de twee delen van het produkt gebruikt men wel de benamingen "meest signifikante deel", en " minst signifikante deel ", die dan respectievelijk in A en B staan. Ook worden vaak de benamingen " kop " en " staart " gebruikt voor de delen in A en B. Het teken van het produkt is automatisch het goede teken, volgens de normale regels dus + maal+ is + ; - maal - is + en tenslotte - maal + is -. Hoewel na vermenigvuldiging A en B samen het produkt vormen, blijven het in een bepaald opzicht toch 2 registers en zijn daardoor elk voorzien van een teken. Na de vermenigvuldiging is dit teken voor A en B in ieder geval gelijk. De machine voert de berekening altijd op dezelfde manier uit, d.w.z. de cijfers van het produkt komen altijd op dezelfde plaats terecht, ongeacht welke cijfers dit zijn. Het zijn ook altijd 22 cijfers, waarbij vooraan, dus links, wel een aantal nullen kunnen voorkomen, We zeggen dan " het produkt is rechts aangesloten ".

Voorbeelden :
Het produkt van + 90000000000 + 90000000000. wordt
+ 81000000000 + 00000000000
A B

Het produkt van + 90000000 en + 90000000 wordt
+ 00000081000 + 00000000000
A B

Het produkt van + 900 en + 900 wordt
+ 00000000000 + 00000810000
A B

In het laatste geval staat het produkt dus eigenlijk compleet in B, omdat de kop nu 0 is geworden. Dit gebeurt altijd wanneer het produkt in werkelijkheid uit niet meer dan 11 cijfers bestaat. Dit is overigens een veel voorkomend geval. Denken we aan vermenigvuldiging van de prijs per eenheid van een artikel met de aanwezige voorraad. Wanneer de prijs per eenheid is uitgedrukt in centen, terwijl deze prijs altijd kleiner is dan *f*. 10.000,-- dan bestaat het eerste getal dus max. uit 6 cijfers. Weten we bijvoorbeeld, dat de voorraad van enig artikel altijd minder dan 100000 is, dan bestaat het tweede getal dus max. uit 5 cijfers. Het produkt bestaat dan max. uit 11 cijfers en komt geheel in B terecht. Na afloop van de vermenigvuldiging is de inhoud van de A-accumulator dus niet van belang. Wel moeten we bedenken, dat een getal dat voor de uitvoering van de vermenigvuldiging nog in A stond, na afloop geheel verloren is. Zou dat dus later nog nodig zijn geweest dan moeten we het voor het uitvoeren van de vermenigvuldigingsopdracht eerst maar ergens opbergen in het geheugen.

DELEN

Bij deling vindt precies het omgekeerde van de vermenigvuldiging plaats. Voor uitvoering van de deelopdracht staat in A en B samen een getal van "dubbele-lengte" rechts aangesloten. De deelopdracht zorgt, dat dit getal gedeeld wordt door een getal uit het geheugen. Na afloop van de deling staat het quotient in B en de rest in A. Ook hier kunnen we weer de situatie ontmoeten, dat we met kleinere getallen werken. Wanneer het deeltal kleiner is dan 11 cijfers, dan moeten we er om denken dat dit getal voor de deling rechts in B staat. Bovendien moeten we er nu aan denken eerst A "schoon" te maken. d.w.z. zorgen dat in A allemaal nullen komen,want het blijft tenslotte de kop van het deeltal. Verder dient er bij een deling nog rekening gehouden te worden met het volgende.
Wanneer we de kop van het deeltal als afzonderlijk getal beschouwen , moet dit in ieder geval kleiner zijn dan de deler. Als dit niet zo is, dus wanneer de kop van het deeltal groter is dan de deler, dan is het werkelijke deeltal dus groter dan 10^{11} keer de deler. Maar dat betekent

dat het quotient groter dan 10^{11} wordt en dat past niet meer in een woord. Na afloop van de deling moet het quotient echter in B kunnen staan en dus in één woord passen.

DE TEKENS

Ten aanzien van de tekens geldt het volgende : het quotient heeft het gebruikelijke teken, de rest heeft altijd het teken van het deeltal.

De tekenregels volgen ook uit het volgende voorbeeld :

deeltal	deler	quotient	rest
+ 14	+ 4	+ 3	+ 2
+ 14	- 4	- 3	+ 2
- 14	+ 4	- 3	- 2
- 14	- 4	+ 3	- 2

We hebben gezien, dat bij vermenigvuldiging en deling A en B gezamelijk optreden. We zullen later nog andere opdrachten ontmoeten waardoor, in A en B samen, een getal van dubbele-lengte kan ontstaan. Daarbij kan zich het volgende geval voordoen. Stel dat in A en B reeds aanwezig is het volgende dubbele-lengte-getal

+ 8239852803735954782164

Dit staat er dan als

+ 82398528037	+ 35954782164
A	B

Nemen we verder aan dat ergens in het geheugen het volgende dubbele-lengte-getal staat

- 2071573891474285168532

Noodzakelijkerwijs staat dit dan in 2 woorden met de kop - 20715738914 in één woord en de staart - 74285168532 in een ander woord.
Als we dit dubbele-lengte-getal willen optellen bij het getal in A en B, dan kan dat door de staart bij B op te tellen en de kop bij A. Voeren we dit uit dan ontstaat in A en B echter het volgende resultaat.

+ 61682789123	- 38330386368
A	B

Kop en staart hebben nu niet meer gelijk teken. Rekenkundig is het antwoord nog steeds goed. Het is hetzelfde alsof we het getal 83, waarmee we bedoelen 80 + 3, schrijven als 90 - 7. Door gelijkmaking van de tekens in A en B, waarbij een eventuele overdracht of leen van B naar A in rekening wordt gebracht, kunnen we weer de normale voorstellingswijze bereiken.
In het gegeven voorbeeld ontstaan dan

+ 61682789122	+ 61669613632
A	B

Bij de uitvoering van de deling zouden moeilijkheden kunnen

ontstaan als van het deeltal in A en B kop en staart verschillende tekens zouden hebben. Daarom is in de SERA de deelopdracht zo gemaakt, dat vóór het uitvoeren van de echte deling eerst A en B tekengelijk worden gemaakt. Een gelijke bewerking vindt vooraf plaats bij iedere opdracht, die bij de uitvoering hinder zou kunnen ondervinden van ongelijke tekens van kop en staart. In al deze gevallen gelden de volgende conventies. Als de kop niet nul is, dan wordt het teken van de staart gelijk gemaakt aan dat van de kop ; de noodzakelijke overdracht of leen wordt daarbij in A verrekend. Is de kop wel nul, maar de staart niet dan wordt de kop + 0 of - 0 gemaakt overeenkomstig het teken van de staart. Is de kop nul en de staart ook, dan worden beiden in ieder geval + 0 gemaakt. Overige eigenschappen van het rekenorgaan komen aan de orde, wanneer de verschillende typen opdrachten nader omschreven worden.

DRIJVENDE KOMMA REPRESENTATIE

Bij alle voorgaande beschouwingen over rekenen in een computer werd stilzwijgend aangenomen dat alle getallen ook gehele getallen waren. Vooral ook voor administratieve problemen is dit volkomen reëel. Natuurlijk kunnen daarbij wel getallen voorkomen waarin een decimale komma gebruikt wordt. Denk maar aan geldbedragen van guldens met 2 cijfers achter de komma. Door in zo'n geval dan eigenlijk te rekenen in centen is men weer terug op een geheel getal. Voorzover men resultaten netjes afgedrukt wil hebben met een komma er in kan men dit altijd verzorgen door eenvoudig op de juiste plaats een komma te laten afdrukken. Men heeft er dan echter in feite niet mee gerekend. Het werken met dergelijke getallen noemt men in de rekenmachine techniek vaak de *VASTE KOMMA BEWERKINGEN*. De kommaplaats ligt n.l. vast en heeft geen invloed op het resultaat. Enerzijds werkt men op deze wijze volkomen nauwkeurig want geen enkel cijfer gaat verloren, anderzijds kunnen wel eens grote getallen ontstaan, wanneer bijvoorbeeld een aantal getallen met elkaar vermenigvuldigd moeten worden. Voor administratieve problemen is dit in de regel geen probleem, de uit te voeren bewerkingen zijn slechts eenvoudig zodat praktisch geen "grote" getallen ontstaan en wanneer ze eventueel ontstaan dan hebben veelal ook alle cijfers betekenis. Deze situatie is wel wat anders bij wetenschappelijke berekeningen. Hier kunnen de bewerkingen zeer gecompliceerd zijn. Daarbij is een beperkte nauwkeurigheid tot bijv. 7 of 8 cijfers geen bezwaar mits men maar wel getallen van verschillende grootorde kan hanteren.
Als mechanisme is daarvoor ingevoerd het begrip drijvende komma *(floating point)*. In feite wordt daarbij ieder getal voorgesteld door 2 afzonderlijke grootheden. De eerste grootheid geeft dan de 1e belangrijke, zogenaamde *significante* cijfers aan.
De tweede grootheid bepaalt de plaats waar de decimale komma thuis hoort ten opzichte van de gegeven cijfers van

de eerste grootheid. Deze laatste grootheid bepaalt dus eigenlijk de grootorde van het gehele getal.
Er is geen algemene definitie te geven omdat voor verschillende machines verschillende representaties gekozen zijn.
Een veel gebruikte vorm en ook bovendien een der oudste voorstellingswijzen berust op de formule $g=a \times 10^b$.
Hierin stelt g een willekeurig getal voor. Daarbij wordt g dus volledig bepaald door de twee grootheden a en b, waarbij aan a en b wel bepaalde restricties verbonden zijn.
Bijvoorbeeld als volgt :

$$0{,}1 \leqslant |a| < 1$$

Het getal a is dus een echte breuk van het type $\pm$ 0,.... met bovendien de eigenschap dat het eerste cijfer achter de komma niet 0 is, d.w.z. dit eerste cijfer is een "echt" cijfer. Bovendien, afhankelijk van het type machine waarop gewerkt wordt, wordt a slechts gegeven in een beperkt aantal cijfers bijvoorbeeld 8 of 10 of eventueel meer. Voor het getal b geldt de restrictie dat het een geheel getal is dat zowel positief als negatief mag zijn en bovendien in de praktijk altijd beperkt tot een zekere max. grootte bijvoorbeeld niet meer dan 4 of 5 cijfers.
Op deze wijze kunnen we echter, zij het dan met beperkte relatieve nauwkeurigheid een enorme range van getallen bestrijken. Laten we als voorbeeld nemen dat a gegeven wordt in 8 significante cijfers en b in max. 3 cijfers.
Het kleinste positieve getal is dan gegeven door a=+0,1 en b=-1000,

dus dan $g=+0{,}1 \times 10^{-1000}$
Voor bijna alle technische toepassingen betekent dit praktisch 0. Anderzijds het grootste getal wordt bepaald door a=+0,99999999 en b=+10000
dus $g=+0{,}99999999 \times 10^{+1000}$
Een zelfde getalbereik geldt dan ook voor de negatieve getallen. Men duidt het getal "a" vaak aan met de naam "mantisse" en "b" met de naam "exponent". De speciale voorwaarde voor "a" dat de absolute waarde ligt tussen 0,1 en 1 geeft men wel aan met de benaming "genormaliseerd" en een getal "g" dat voldoet aan bovengenoemde eisen noemt men wel een "genormaliseerd drijvend komma getal". Laat men alleen de eis $0{,}1 \leqslant |a| < 1$ vallen, maar handhaaft men de verdere voorstelling dan spreekt men wel van " niet-genormaliseerd" of "ongenormaliseerd drijvend komma getal " De laatste vorm kan soms als tussenresultaat ontstaan, zoals ook dadelijk uit voorbeelden zal blijken. De bewerking van een getal om het weer in genormaliseerde vorm te brengen wordt aangeduid met "normaliseren"

Wanneer in de literatuur of handboeken van computers zonder meer gesproken wordt over drijvende komma getallen en bewerkingen daarmee, dan bedoelt men in de regel de genormaliseerde vorm.

Allereerst enkele voorbeelden met naast elkaar de gebruikelijke schrijfwijze en daarnaast de genormaliseerde drijvende komma vorm; daarbij worden mantisse en exponent zonder meer naast elkaar geschreven met een kleine ruimte er tussen, zoals ook veelal gebruikelijk is bij standaarduitvoer op machines, die voor drijvende komma zijn ingericht.

normaal	drijvend
34	+ 0,34000000 + 002
34,72	+ 0,34720000 + 002
7298,5432	+ 0,72985432 + 004
0,0037654	+ 0,37654000 - 002
-0,00000215	- 0,21500000 - 005
+ 7129854321574	+ 0,71298543 + 013

Uit de definitie volgt vanzelf hoe verschillende rekenbewerkingen met drijvende komma getallen moeten worden uitgevoerd. Bij vermenigvuldigen worden de mantissen vermenigvuldigd en afgerond op het juiste gefixeerde aantal cijfers terwijl de exponenten worden opgeteld. Bijvoorbeeld :
7,65 x 83,7 = 640,305
verloopt nu als volgt :

+ 0,76500000 + 001 x + 0,83700000 + 002 = 0,64030500 + 003

In sommige gevallen zal aanvankelijk het tussenprodukt van de mantissen niet aan de eisen voldoen, bijvoorbeeld :
2,36 x 3,14 = 7,4104
Uitgaande van + 0,23600000 + 001 en
+ 0,31400000 + 001
ontstaat aanvankelijk
+ 0,0741040000000000 + 002
d.w.z. een mantisse van 16 cijfers. Deze begint nu echter met een nul direkt achter de komma. Daarom wordt "genormaliseerd" d.w.z. alle cijfers worden 1 plaats naar links geschoven , daarna vindt afronding plaats op 8 cijfers achter de komma en in verband met deze verschuiving wordt de exponent herzien zodat in dat geval ontstaat
+ 0,74104000 + 001

Bij optellen worden eerst de exponenten met elkaar vergeleken. Blijken de exponenten 9 of meer van elkaar te verschillen dan is het ene getal binnen de gegeven nauwkeurigheid van 8 cijfers dus volledig verwaarloosbaar t.o.v. de andere. Is het verschil kleiner, dan wordt het kleinste getal zoveel naar rechts geschoven onder de andere totdat de mantissen opgeteld kunnen worden. Daarna wordt gecontroleerd of eventueel genormaliseerd moet worden en vindt afronding op 8 cijfers plaats.

Enkele voorbeelden :

```
                    37,2 + 3,59 = 40,79
    37,2 ------->   +0,37200000  +002
     3,59------->   +0,35900000  +001

                    +0,37200000  +002
                    +0,03590000 ++002
                    +0,40790000  +002
        37215       +0,0038214 = 37215,0038124

        37215       +0,37215000  +005
     0,0038214      +0,38214000  -002

                    +0,37215000           +005
                    +0,0000000382140000,  +005
                    +0,3721500382140000+  +005
                    +0,37215004  +005
```

Er kan ook tijdelijk overloop optreden.

7654 + 9214 = 16868

```
7654 +0,76540000  +004
9214 +0,92140000  +004

 +0,76540000 +004
 +0,92140000,+004
 +1,69690000+ +004
 +0,16868000 +005
```

Bij het aftrekken van twee getallen of het optellen van een positief en een negatief getal kan een situatie optreden waarbij normaliseren over een aantal cijferplaatsen noodzakelijk is.

51492578 - 51492536 = 42

```
51492578                +0,51492578  +008
                        +0,51492536- +008
                        +0,00000042  +008
                        +0,42000000  +002
```

Om weer een genormaliseerde mantisse te verkrijgen wordt in dit geval over 6 plaatsen naar links geschoven en de exponent met 6 verminderd. Daarbij wordt de nieuwe mantisse noodgedwongen rechts met 6 nullen aangevuld. Deze hebben echter geen reële betekenis voor de nauwkeurigheid. Vooral wanneer een dergelijk resultaat weer gebruikt wordt voor verdere berekeningen bijvoorbeeld : vermenigvuldigd met een ander getal dan ontstaan resultaten met ogenschijnlijk weer 8 "echte" cijfers maar in feite zijn dan alleen de eerste twee cijfers betrouwbaar. Men spreekt in zo'n geval wel van schijnnauwkeurigheid. Bij het opstellen van een berekeningsschema, waarbij men gebruik denkt te maken

van drijvende komma dient men met dit effect rekening te houden.
Nemen we als voorbeeld aan dat als deel van een berekening moet worden berekend a^2-b^2 waarbij a en b drijvende komma getallen voorstellen, die in de praktijk nog alle mogelijke waarden kunnen hebben. Zolang a en b redelijk van elkaar verschillen zal de uitkomst ook vrij nauwkeurig zijn. Naarmate a en b echter minder verschillen wordt de onnauwkeurigheid steeds groter.
Stel a = 78549215 en b = 78549214

In drijvende komma vorm
a = +0,78549215 +008
b = +0,78549214 +008

Berekening van a^2 geeft aanvankelijk

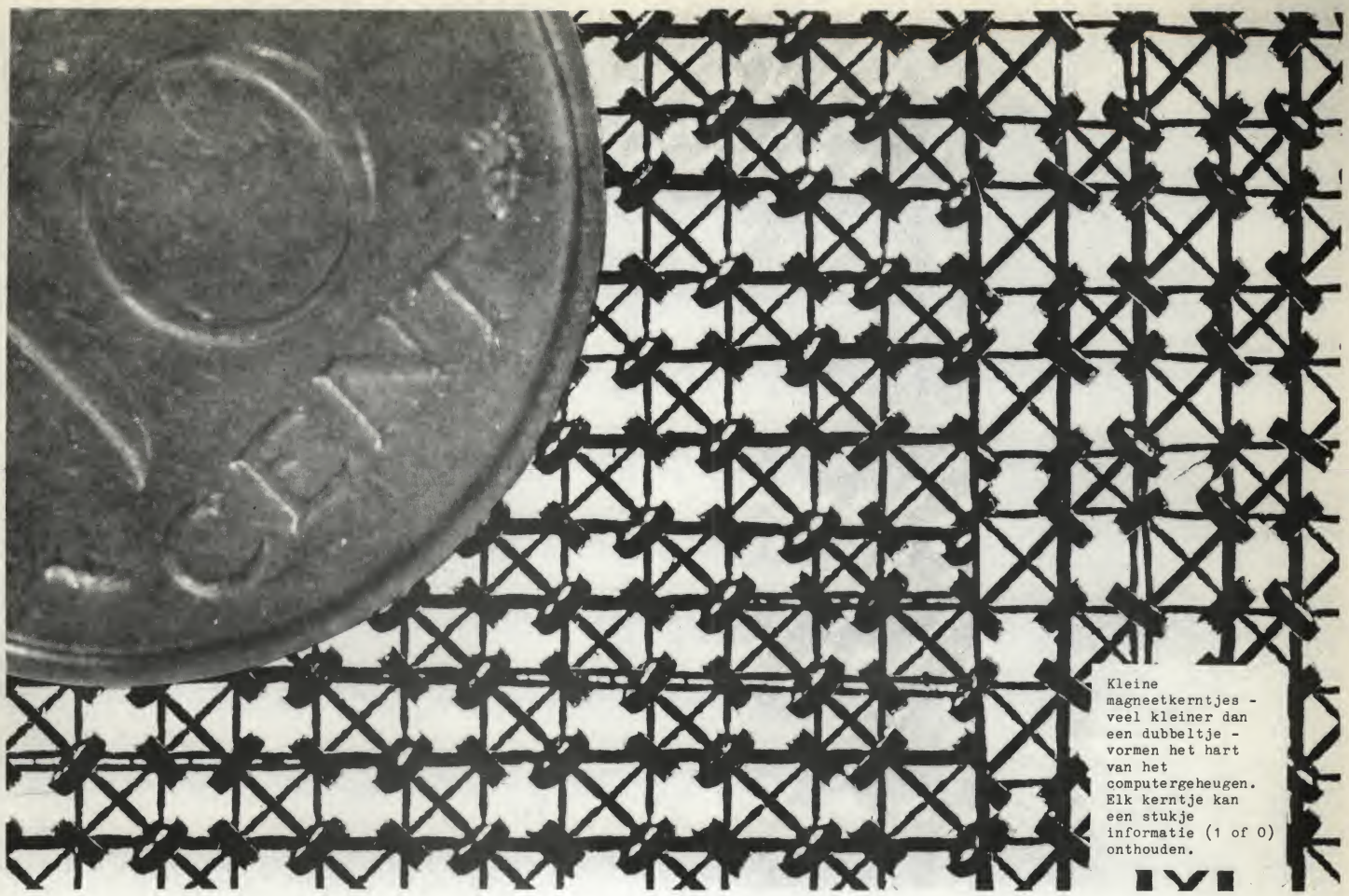
a^2 = +0,6169979177116225 +016
en na afronding op normale lengte

a^2 = +0,61699792 +016

zo ook b^2 = +0,6169979020017796 +016
en na afronding

b^2 = +0,61699790 +016

Bepalen we nu a^2-b^2 dan ontstaat
+0,00000002 +016
en na normalisatie
+0,20000000 +009
De exacte uitkomst zou in dit geval geweest zijn
157099429
De uitkomst is dus nog slechts in één cijfer nauwkeurig.
Deze situatie kan hier vermeden worden door de berekeningswijze te herzien. We veranderen eerst de formule
$a^2 - b^2$ in (a+b)x(a-b).
Algebraïsch is deze formule identiek, maar voor de berekening in drijvende komma maakt het een groot verschil.
Allereerst a+b levert aanvankelijk
+1,57098429 +008
na normalisatie en afronding
+0,15709843 +009
Daarna a-b geeft
+0,000000001 +008
Na normalisatie +0,10000000 +001
Vermenigvuldiging van deze twee resultaten geeft aanvankelijk
+0,0157098430000000 +010
Na normalisatie en afronding ontstaat dan +0,15709843 +009.
Dit is met een cijfernauwkeurigheid van 8 cijfers het best denkbare resultaat. Helaas is in de praktijk de oplossing niet altijd zo eenvoudig te vinden als hier en moet men



Kleine magneetkerntjes - veel kleiner dan een dubbeltje - vormen het hart van het computergeheugen. Elk kerntje kan een stukje informatie (1 of 0) onthouden.

"O" CONDITIE	"1" CONDITIE
	IBM PONSKAART
	RELAIS OF SCHAKELAAR
	BUIS OF TRANSISTOR
Uit Aan	GLOEILAMP Uit Aan

Magneetkernen, uiterst kleine ferrietkraaltjes in een netwerk van fijne stroomdraden, nog steeds het meest toegepaste elektronische geheugen

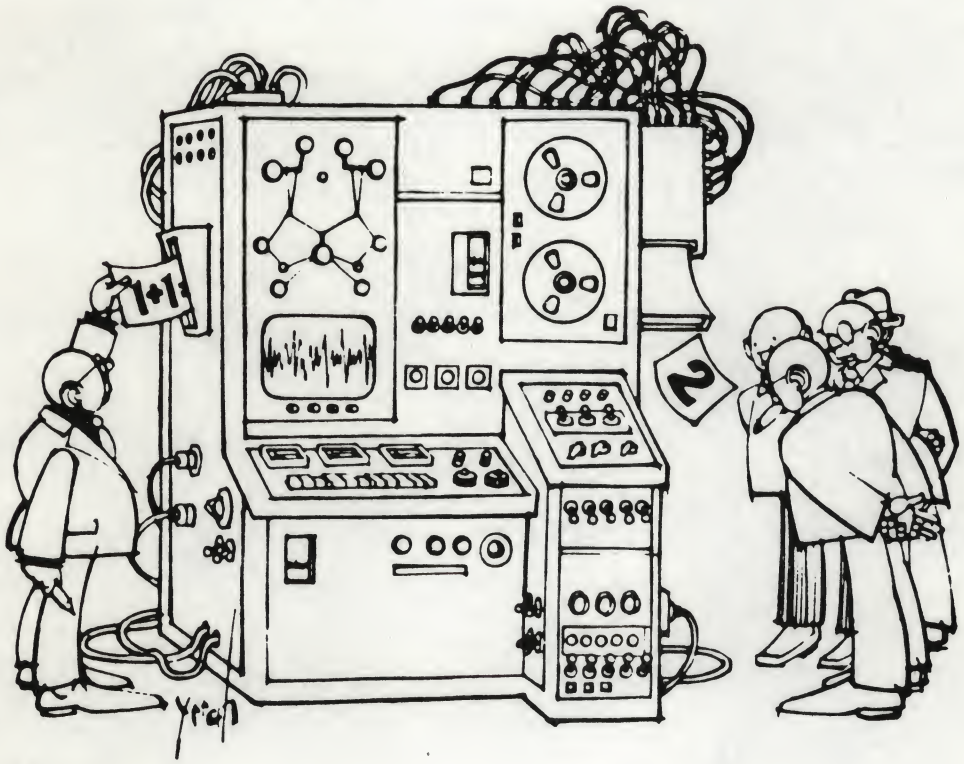
berekeningswijzen soms drastisch herzien om bovengenoemde moeilijkheden te omzeilen.
Is een dergelijke oplossing niet of alleen met zeer veel moeite te vinden dan kan een oplossing ook zijn om een gedeelte van de berekening toch in vaste komma uit te voeren en daarbij dan gebruik te maken van multilengte nauwkeurigheid.

TENSLOTTE ENKELE ALGEMENE OPMERKINGEN

Uit de voorbeelden blijkt dat iedere bewerking met drijvende komma getallen op zichzelf ontleed kan worden in een aantal deelbewerkingen, die ieder voor zich met normale opdrachten in een rekenmachine kunnen worden uitgevoerd. Bergen we bijvoorbeeld ieder drijvend komma getal op in het geheugen op 2 achtereenvolgende plaatsen n.l. 1 voor de mantisse en 1 voor de exponent, dan kunnen we programma's samenstellen, die steeds twee van dergelijke getallen vermenigvuldigen, delen, optellen of aftrekken. Dit was de manier waarop dit bij de eerste machines werd gedaan. Bij latere machines en speciaal die voor wetenschappelijk gebruik, bouwde men speciale opdrachten in de machine, die de taak van dergelijke programma's konden overnemen.
Een volgende stap was om geheugenruimte te besparen, en niet steeds 2 woorden nodig te hebben voor ieder drijvend komma getal, dat mantisse en exponent werden samengepakt in één woord. Weliswaar moeten dan steeds voor iedere bewerking mantisse en exponent dan weer van elkaar gescheiden worden, maar wanneer dergelijke opdrachten toch worden ingebouwd, vormt deze wijze van werken geen ernstige complikatie.
Uit de voorbeelden is ook gebleken dat de normalisatie bewerking relatief vaak optreedt. Deze bewerking komt in principe neer op decimaal schuiven. Hebben we nu te maken met een machine die in principe toch decimaal werkt dan is dit geen bezwaar. Anders ligt de situatie wanneer de machine zelf normaal zuiver binair werkt. Decimaal naar links schuiven betekent dan steeds compleet vermenigvuldigen met 10 en nog ernstiger naar rechts schuiven betekent delen door 10. In dergelijke gevallen past men vaak binair drijvende komma getallen toe. In zo'n geval wordt dan een getal g bepaald door: $g = a \times 2^b$.
met $\frac{1}{2} \leqslant |a| < 1$ en $|b|$ geheel en kleiner dan een zeker getal. Daarmee is normaliseren weer terug gebracht tot schuiven. Weliswaar ontstaat dan weer het nadeel dat t.b.v. uitvoer toch weer omgerekend moet worden tot decimaal drijvend, omdat de binair drijvende komma voorstelling ons mensen weinig aanspreekt.
Een ander punt van overweging is dat men voor de exponent b die nu slechts een macht van 2 aangeeft relatief grote getallen moet toelaten om toch nog een relatief groot bereik van alle drijvende komma getallen te behouden. Dit is dan weer de oorzaak dat men soms als grondtal weer een

macht van 2 kiest bijvoorbeeld 8.

Dan wordt $g=ax8^b$ met $\frac{1}{8} \leqslant |a| < 1$ en $|b|$ weer geheel en begrensd.
In zo'n geval betekent normaliseren een aantal malen steeds over 3 bits schuiven omdat $8=2^3$
Ook in deze gevallen wordt dan meestal weer mantisse en exponent in een woord samengepakt.



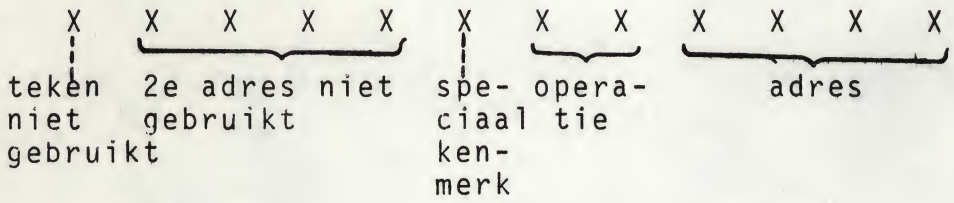
Toch is het binaire talstelsel soms eenvoudiger dan men denkt.

5. HET BESTURINGSORGAAN VAN DE SERA

Dit orgaan bevat allereerst een register ter grootte van een geheel SERA woord, waarbij dan echter alleen een numerieke samenstelling betekenis heeft. Dit register noemen we het uitvoeringsregister en kan dus bevatten 11 decimale cijfers alsmede een teken. Het besturingsorgaan bevat nog een tweede register, dat alleen een decimaal getal van max. 4 cijfers kan bevatten. Dit register noemen we de opdrachtenteller.
Het besturingsorgaan in zijn geheel moet er nu zorg voor dragen, dat opgegeven opdrachten, die in het geheugen staan ook uitgevoerd worden. Daarom wordt steeds een woord uit het geheugen gehaald en in het uitvoeringsregister geplaatst. Welk woord uit het geheugen wordt gehaald, wordt bepaald

door de opdrachtenteller. Deze bevat een getal van 4 cijfers hetgeen juist een adres aangeeft. Het besturingsorgaan werkt in 2 fasen. In de eerste fase wordt de inhoud van het woord in het geheugen, welk adres juist wordt aangegeven door het getal in de opdrachtenteller, naar het uitvoeringsregister gebracht.
In de tweede fase wordt het "getal", dat nu in het uitvoeringsregister staat, beschouwd als opdracht en als zodanig uitgevoerd. Tijdens de uitvoering van deze opdracht wordt tevens het getal dat in de opdrachtenteller staat met 1 vermeerderd. Hierna volgt weer de eerste fase, waarbij nu automatisch de volgende opdracht wordt gehaald enz.

Voor het programmeren is van belang op welke manier het uitvoeringsorgaan een gegeven decimaal getal als opdracht interpreteert. Dit gebeurt als volgt.
De twaalf tetraden waaruit een woord bestaat, worden niet meer beschouwd als één decimaal getal van 11 cijfers met teken, maar gesplitst in een aantal delen met ieder een eigen betekenis.



Van rechts af gerekend vormen de eerste vier cijfers een getal, dat een adres aangeeft. Dit gedeelte noemen we het adresgedeelte van een woord.
Daarna volgt het operatiegedeelte bestaande uit 2 cijfers. Vervolgens komt een cijfer, dat een opdracht een speciaal kenmerk kan geven. Als dit cijfer 0 is, betekent het dat geen kenmerk aanwezig is. Voorlopig zullen we aannemen, dat dit bij alle te bespreken opdrachten het geval is. Pas in latere hoofdstukken zullen we toepassingen ontmoeten, waarbij dit kenmerk pas betekenis krijgt. Links van het kenmerk blijven nog 4 cijferplaatsen over. Deze hebben voor de meeste opdrachten geen betekenis, mogen daarom met willekeurige cijfers gevuld zijn. Bij het uitvoeren van deze opdrachten laat het uitvoeringsorgaan deze cijfers ook geheel buiten beschouwing.

In enkele bijzondere opdrachten hebben deze 4 plaatsen wel betekenis en geven dan een 2e adres aan. Tenslotte de tekenpositie; deze is voor de opdracht niet van belang en wordt niet gebruikt.
Voorlopig hebben we dus alleen te maken met de rechtse 6 cijferplaatsen. Het operatiegedeelte geeft daarbij, als

getal van 2 cijfers, aan welk type opdracht moet worden uitgevoerd. Het adresgedeelte geeft aan een adres in het geheugen, waar we de operand kunnen vinden. In het algemeen is de operand dus een getal, dat in het geheugen staat en waarvan het adres wordt aangegeven in de opdracht. Omdat de operatie bestaat uit een getal van 2 cijfers kunnen we dus 100 verschillende type opdrachten hebben. Niet alle mogelijkheden zijn echter gebruikt; bepaalde getallen stellen dus een zinloze operatie voor. Mocht door een of andere omstandigheid toch een dergelijke combinatie in het uitvoeringsregister terecht komen dan gebeurt er niets en wordt gewoon verder gegaan. De opdracht reageert dan als "dummy opdracht".

We zullen het voorgaande toelichten aan de hand van een eenvoudig voorbeeld. Daartoe nemen we het reeds gebruikte voorbeeld van optellen van 2 getallen in de B-accumulator. Voor enkele opdrachten moeten we nu allereerst de opdrachtcode weten.

Dit zijn : 03 haal een getal in de B-accumulator
12 tel een getal in B op
05 breng een getal vanuit B naar geheugen.

Voor de overzichtelijkheid zullen we ook nog 2 echte getallen nemen, die opgeteld moeten worden. Stel deze getallen zijn + 365 en + 7893. Deze twee getallen moeten ergens in het geheugen staan. Nemen we aan op de plaatsen 405 en 6445. Dat betekent : (405) = + 365
(6445) = + 7893
We wensen verder dat het resultaat van de optelling terecht komt in woord 9000. Het benodigde stukje programma moet ook nog ergens in het geheugen staan. Neem aan vanaf adres 100.
We schrijven dit stukje programma als volgt :
Voor de kantlijn de adressen, waar de *OPDRACHTEN OF INSTRUKTIES* staan.

Achter de kantlijn de instruktie zelf.

100	+ 00000030405
101	+ 00000126445
102	+ 00000059000
103	

Zouden we er technisch voor zorgen, dat de opdrachtenteller op 100 komt te staan en laten we de machine verder lopen dan gebeurt het volgende.
De inhoud van 100 wordt naar het uitvoeringsregister gebracht. De opdracht wordt uitgevoerd. Dat is in dit geval: haal het getal, dat in 405 staat naar de B-accumulator d.w.z. + 365 gaat naar B. De opdrachtenteller wordt nu 101. De inhoud van 101 gaat naar het uitvoeringsregister. Uitvoering opdracht : tel op in B het getal dat in 6445 staat. In B ontstaat dus nu het getal + 8258. De opdrachtenteller wordt

102.
De inhoud van 102 gaat naar het uitvoeringsregister. Uitvoering opdracht : het getal in B wordt gebracht naar plaats 9000, dus + 8258 gaat naar 9000, de opdrachtenteller wordt 103 enz.
De rest kunnen we niet beschrijven, omdat we niet hebben aangegeven wat in 103 en volgende plaatsen staat.

INVOERPROGRAMMA
Bij dit voorbeeld hebben we aangenomen, dat alle noodzakelijke gegevens reeds in het geheugen aanwezig zijn en bovendien de opdrachtenteller in de goede startwaarde staat. Daarmee hebben we de moeilijkheid, om de gegevens in de machine te krijgen, omzeild. Dat inbrengen van gegevens vindt plaats via het invoerorgaan. Dat is op zich geen moeilijkheid, omdat in de opdrachtencode ook enkele combinaties voorkomen die betrekking hebben op het invoerorgaan. Het besturingsorgaan kan echter alleen instrukties uit het geheugen ophalen. Het programma, dat de instrukties bevat voor het inlezen van het door ons aangegeven programma, moet dus reeds in het geheugen aanwezig zijn. Redeneert men op deze manier verder, dan komt men in een vicieuse cirkel terecht, want hoe komt dan dit programma in het geheugen? Deze moeilijkheid bestaat inderdaad en heeft men als volgt opgelost. Er wordt voor gezorgd, dat een bepaald basisprogramma altijd in het geheugen aanwezig is. Alleen door technische ingreep kan men dit programma wijzigen. Bij normaal gebruik van de machine zijn de geheugenplaatsen waar dit bassisprogramma staat gesperd voor schrijven en kan men dit programma dan ook niet verstoren. Verder is technisch de mogelijkheid aanwezig om dit programma te starten. Dit gebeurt eenvoudig door de opdrachtenteller kunstmatig op het beginadres van het programma te zetten. Dit basisprogramma staat bekend onder de naam *INVOERPROGRAMMA*.
Met behulp van dit invoerprogramma kunnen we een programma inlezen. De gang van zaken is, dat we eerst op papier een programma schrijven. Dit geschreven programma wordt daarna geponst in bijvoorbeeld ponskaarten. Vervolgens worden deze kaarten in de ponskaartenlezer van de machine gelegd. Nu starten we het invoerprogramma. Dit invoerprogramma bestaat onder andere uit een aantal instrukties, die steeds de opdracht "lees een kaart" uitvoeren. Verder zorgt het invoerprogramma er dan voor dat de "gelezen" instrukties op de juiste plaats in het geheugen komen. Nadat alle kaarten op deze manier "ingelezen" zijn,stopt de machine weer, Op dit moment staat het nieuwe programma klaar in het geheugen, maar het heeft nog niet gewerkt. Stel dat het een programma voor loonberekening is. Om deze berekeningen te kunnen uitvoeren zijn per werknemer verschillende gegevens nodig zoals brutoloon, voor de belastingberekening al of niet gehuwd, aantal kinderen enz.

Ook deze gegevens zijn in ponskaarten vastgelegd. Deze stapel ponskaarten brengen we thans in de ponskaartenlezer en daarna starten we het loonberekeningsprogramma, zoals dat in het geheugen staat. Ook dit programma zal dus instrukties van het type "lees een kaart" bevatten en door deze instrukties worden de kaarten met werknemersgegevens ingelezen. We moeten dus goed onderscheiden: een nieuw programma wordt ingelezen met behulp van het altijd aanwezige invoerprogramma, de gegevens waarmee gerekend moet worden, worden ingelezen door het programma zelf.
De functie van het invoerprogramma is dus alleen de getallen, die instrukties voorstellen, in te lezen vanaf ponskaarten of ponsband en deze getallen in het geheugen te plaatsen in opeenvolgende woorden. In dat geval is het invoerprogramma zelf bijzonder eenvoudig. Niet eenvoudig is dan het schrijven van programma's. Dit moet volledig plaats vinden op dezelfde manier zoals in het gegeven voorbeeld. Men heeft daarom talloze middelen ter hand genomen om dit programmeren eenvoudiger te maken. Het resultaat daarvan is, dat men een programma op eenvoudige manier en tegelijkertijd veel overzichtelijker kan schrijven. Wel moet men altijd bepaalde conventies in acht nemen, die er voor zorgen dat er een eenvoudige correspondentie blijft bestaan met het programma zoals het in het geheugen van de machine moet komen.
De noodzakelijke omzetting of vertaling laten we uitvoeren door het invoerprogramma. Dat is mogelijk omdat genoemde correspondentie berust op volkomen logische regels. De konsequentie is wel, dat dit invoerprogramma daardoor aanzienlijk gecompliceerd kan worden. Dit is niet zo bezwaarlijk, omdat het invoerprogramma tenslotte maar één keer samengesteld behoeft te worden, terwijl het gemak dat hiermee bereikt wordt, zich uitstrekt over alle programma's, die daarna nog gemaakt worden. Ook voor de SERA is een invoerprogramma met veel faciliteiten aanwezig. De conventies hiervoor zullen we in de loop van deze kursus leren kennen.

MNEMOTECHNISCHE CODE

De eerste mogelijkheid, die het invoerprogramma biedt, is, dat we op papier niet meer behoeven te werken met de 2-cijferige operatiecode zoals die intern voorkomt. Iedere operatie wordt nu voorgesteld door een combinatie van 3 letters, die zo gekozen zijn dat zij zo goed mogelijk de werking van de operatie aangeven. Men noemt dit wel een mnemotechnische code. Iedere instruktie bevat ook een adres. In de beschrijving geven we dit meestal aan met de letter n
Voorbeeld :

Benaming	Werking	Omschrijving
HPA n	(n) ⟶ A	Haal Positief in A

De operatiecode is dus in dit geval HPA. Indien hierbij be-

hoort een adres n, dan zorgt bij de uitvoering deze instruktie er voor, dat de inhoud van n gebracht wordt naar de A-accumulator. Dat laatste geven we symbolisch weer door het pijltje.

We geven thans een overzicht van de belangrijkste haal-breng- en rekeninstrukties.

Benaming		Werking		Omschrijving
HPA	n	(n) ⟶	A	Haal Positief in A
HNA	n	-(n) ⟶	A	Haal Negatief in A
HPB	n	(n) ⟶	B	Haal Positief in B
HNB	n	-(n) ⟶	B	Haal Negatief in B
BPA	n	(A) ⟶	n	Breng Positief uit A
BNA	n	-(A) ⟶	n	Breng Negatief uit A
BSA	n	(A) ⟶	n daarna	Breng Schoon uit A
		0 ⟶	A	
BPB	n	(B) ⟶	n	Breng Positief uit B
BNB	n	-(B) ⟶	n	Breng Negatief uit B
BSB	n	(B) ⟶	n daarna	Breng Schoon uit B
		0 ⟶	B	
OPA	n	(A) + (n) ⟶	A	Optellen in A
AFA	n	(A) - (n) ⟶	A	Aftrekken in A
OPB	n	(B) + (n) ⟶	B	Optellen in B
AFB	n	(B) - (n) ⟶	B	Aftrekken in B
VMG	n	(B) x (n) ⟶	AB	Vermenigvuldigen
DLN	n	(AB) / (n) ⟶	A en B	Delen

waarbij (A) = rest
(B) = quotient

TOELICHTING BIJ DEZE INSTRUKTIES

Overal waar dit niet expliciet is aangegeven, blijft de inhoud van een register of een woord onveranderd. Dus (A)⟶ n betekent de inhoud van A gaat naar n; hiervan gaat dus de inhoud verloren, maar in A blijft staan wat stond. In dit geval blijft bovendien B onaangetast. Bij de opdrachten BSA en BSB worden eigenlijk twee handelingen na elkaar verricht, n.l. eerst opbergen vanuit één der accumulatoren naar woord n en daarna wordt de betrokken accumulator schoon gemaakt d.w.z. gevuld met nullen.

Ir. D.H. Wolbers

Hierna geven we enkele voorbeelden van korte stukjes programma voor het uitvoeren van enkele berekeningen. Deze programma's laten we steeds beginnen op adres 100. Verder wordt aangenomen, dat de noodzakelijke getallen a,b,c,d enz. voorzover die nodig zijn, reeds in het geheugen staan vanaf plaats 200, dus (200) = a; (201) = b enz. Het resultaat gaat steeds naar 300.

Allereerst x = a + b.

```
100 |   HPA   200
101 |   OPA   201
102 |   BPA   300
```

We nemen dus de volgende regels in acht. Voor de kantlijn het adres, waar de instrukties moeten komen. Achter de kantlijn eerst de operatie door een 3-letter code, daarachter het adres. Tussen operatie en adres laten we enige ruimte. De ruimte achter het adres kan gebruikt worden voor verklarende tekst, die overigens niet in de machine wordt ingevoerd maar kan dienen om het zoeken van fouten later te vereenvoudigen.

x = a - b

```
100 |   HPA   200          a ⟶ A
101 |   AFA   201    a -   b ⟶ A
102 |   BPA   300          x ⟶ 300
```

Dit geval zou als volgt geprogrammeerd kunnen worden

```
100 |   HNA   201         -b ⟶ A
101 |   OPA   200    a -   b ⟶ A
102 |   BPA   300          x ⟶ 300
```

Een mogelijke uitvoering is zelfs ook

```
100 |   HPA   201              b ⟶ A
101 |   AFA   200        b -   a ⟶ A
102 |   BNA   300 - (b-a) = + x ⟶ 300
```

Deze laatste vorm zou van belang kunnen zijn indien we direkt hierna ook nog de waarde -x nodig zouden hebben. Deze staat nu immers klaar in A

x = a + b - c - d

```
100 |   HPA   200
101 |   OPA   201
102 |   AFA   202
103 |   AFA   203
104 |   BPA   300
```

x = ab + cd

Hierbij nemen we aan, dat de gegeven getallen zo klein zijn dat de produkten nog in enkele-lengte bestaan.

```
100   HPB   200            a ⟶ B
101   VMG   201        a x b ⟶ B
102   BPB   300        a x b ⟶ 200
103   HPB   202            c ⟶ B
104   VMG   203        c x d ⟶ B
105   OPB   300   axb + cxd ⟶ B
106   BPB   300            x ⟶ 300
```

We merken op dat woord 300 ook gebruikt is om tijdelijk een tussenresultaat in op te slaan.
Nemen we dezelfde voorbeelden x = axb = cxd maar nemen we nu aan dat a,b,c en d getallen zijn van enkele lengte, maar wel zo groot, dat het produkt groter wordt dan 11 cijfers. Dit betekent dat we in dubbele lengte moeten optellen, waarbij een overloop kan ontstaan van B naar A. Bij de opdracht OPB gaat een eventuele overdracht echter verloren. Daarom bestaan 2 extra opdrachten, die dit wel doen.

OAB n (B) + (n) ⟶ B met overdracht naar A — Optellen in A en B

AAB n (B) - (n) ⟶ B met overdracht naar A — Aftrekken in A en B

Afhankelijk van de tekens en de grootte van de getallen in B en n kan de overdracht zijn -1, 0 of +1.

Behalve deze overdracht zijn deze opdrachten wat hun werking in B betreft indentiek met OPB an AFB.

Het voorbeeld x = ab + cd in dubbele lengte wordt nu

```
100   HPB   200              a ⟶ B
101   VMG   201          a x b ⟶ AB
102   BPA   300     kop  a x b ⟶ 300
103   BPB   301 staart  a x b ⟶ 301
104   HPB   202              c ⟶ B
105   VMG   203          c x d ⟶ AB
106   OAB   301 optellen staarten
107   OPA   300 optellen koppen
108   BPA   300     kop van x ⟶ 300
109   BPB   301 staart van x ⟶ 301
```

Bij dit stukje programma is wel aangenomen, dat de getallen a,b,c en d niet zo groot zijn, dat de optelling van de produkten een resultaat van 23 cijfers zou geven. Dit geval zou zich bijvoorbeeld voordoen als alle getallen bestaan uit 11 significante cijfers en alle met een 9 beginnen. Nu

Ir. D.H. Wolbers

kan men 23 cijfers niet meer voorstellen in 2 woordlengten, maar moet dan noodgedwongen tot drie dubbele lengte overgaan. Het verwerken van overdrachten kan echter alleen van B naar A. Een overdracht, die links in A ontstaat bij een optelling in A, gaat altijd verloren. Daarom zou in een dergelijk geval het programma als volgt moeten verlopen.

```
100     HPB   200                    a ⟶ B
101     VMG   201               a x b ⟶ AB
102     BPA   300          kop a x b ⟶ 300
103     BPB   301       staart a x b ⟶ 301
104     HPB   202                    c ⟶ B
105     VMG   203               c x d ⟶ AB
106     OAB   301 staarten optellen
107     BPB   302 3e stuk van x ⟶ 302
108     BSA   301 tussenresultaat ⟶ 301; 0 ⟶ A
109     HPB   301 tussenresultaat ⟶ B
110     OAB   300 koppen optellen
111     BPB   301 2e stuk van x ⟶ 301
112     BPA   300 1e stuk van x ⟶ 300
```

De opdracht BSA op plaats 108 is hier zinvol, omdat daarmee een "schone" A-accumulator wordt achtergelaten, waarin, door de opdracht OAB op 110, het eventuele voorste 23e cijfer van x terecht kan komen. Men kan dus, zo men wil, met zoveel cijfers rekenen als met wenst. In de praktijk komt het, vooral bij administratieve problemen, bijna niet voor.

TEKSTVRAGEN

1. *Schets en omschrijf de Sera-standaard instruktievorm.*

2. *Omschrijf tenminste 3 mogelijke adresseringswijzen in een systeem met variabele woordlengte.*

3. *Welke haal/breng instrukties kent U ?*

4. *Welke rekeninstrukties kent U ?*

SPRONGINSTRUKTIES

Wanneer we het programma voor een groot rekenproces zouden opstellen door eenvoudig een aantal stukjes programma, zoals van de vorige voorbeelden achter elkaar te plaatsen dan ontstaat de volgende moeilijkheid. Neem aan dat we beschikken over een geheugen van 10000 plaatsen en we een be-

rekening willen maken m.b.v. 1000 getallen. Voor het programma zijn dan dus nog max. 9000 plaatsen beschikbaar. Voor een moderne elektronische rekenmachine is een gemiddelde uitvoeringstijd per opdracht 50 microsec. zeker haalbaar; vele machines werken zelfs aanzienlijk sneller. De totale rekentijd van 9000 opdrachten is dan 450000 microsec. d.w.z. iets minder dan ½ sec. Dat zou betekenen dat de machine met een max. mogelijke berekening al klaar is nog voor men de startknop reeds weer heeft losgelaten. Op deze wijze zou natuurlijk nooit met rekenmachines gewerkt kunnen worden. De grote kracht van een rekenmachine is echter hierin gelegen dat men de zelfde berekening steeds opnieuw maar weliswaar met andere getalgegevens kan herhalen. Daartoe is het noodzakelijk dat men kan aangeven dat na het uitvoeren van alle benodigde opdrachten voor één berekening opnieuw begonnen moet worden met de eerste opdracht. Denken we terug aan de samenstelling van het besturingsorgaan bestaande uit uitvoeringsregister en opdrachtenteller. Na behandeling van een opdracht wordt de opdrachtenteller met 1 verhoogd waardoor automatisch de volgende opdracht aan de beurt komt. Het gewenste resultaat aan het einde van een berekening kunnen we bereiken door te zorgen dat dan de opdrachtenteller niet met 1 verhoogd wordt, maar precies op dat nummer gezet wordt dat het adres aangeeft van de eerste berekening. Daartoe dient de zogenaamde sprongopdracht.

SAL n Spring altijd naar n

De werking van deze opdracht kunnen we als volgt aangeven, Wanneer een opdracht SAL n in het uitvoeringsregister komt, dan wordt daardoor het getal n in de opdrachtenteller geplaatst. Dit heeft tot gevolg dat de volgende opdracht die opgehaald en daarna uitgevoerd gaat worden diegene zal zijn, die juist op adres n staat. We "springen" in het programma dus als het ware naar plaats n.
De verklaring van "altijd" in deze instruktie is te vinden in het onderscheid met de volgende serie sprongopdrachten. Voor al deze opdrachten geldt gemeenschappelijk het volgende. Bij iedere voorwaardelijke sprongopdracht behoort een bepaalde voorwaarde. Is aan deze voorwaarde voldaan dan reageert een dergelijke opdracht op dezelfde wijze als een SAL opdracht m.a.w. de opdrachtenteller wordt gevuld met het aangegeven adres.
Is echter aan de gestelde voorwaarde niet voldaan dan wordt in feite geen opdracht uitgevoerd, maar wel wordt evenals bij een gewone transport- of rekenopdracht, de opdrachtenteller met 1 verhoogd.Dit betekent dat als volgende opdracht dus uitgevoerd zal worden de opdracht die direkt volgt op de voorwaardelijke sprongopdracht. De voorwaardelijke sprongopdracht biedt in een programma dus de mogelijkheid een vertakking aan te brengen, waarbij de keus afhankelijk

gesteld kan worden van een resultaat, dat in een voorafgaand stuk programma is berekend.

Juist deze keuze mogelijkheid afhankelijk van dynamische resultaten verleent een rekenmachine zijn grote flexibiliteit t.a.v. allerlei soorten problemen. Daarbij geldt echter steeds dat berekeningswijze en de logische gevolgen van bepaalde resultaten tevoren in een programma moeten worden vastgelegd. Doordat een machine daarna echter in een veel sneller en onvermoeibaarder tempo dan de mens deze bewerkingen kan uitvoeren, ontstaat soms de indruk alsof een rekenmachine intelligentie bezit hetgeen zeker niet waar is.
Teneinde op praktische manier met voorwaardelijke sprongopdrachten te kunnen werken heeft men in iedere machine de beschikking over een aantal van dergelijke opdrachten, die ieder voor zich afhankelijk zijn van een andere conditie.

In SERA code beschikken we daartoe over de volgende opdrachten :

SPA	n	Spring naar n als (A) is positief of ± 0
SNA	n	Spring naar n als (A) is negatief
SOA	n	Spring naar n als (A) is 0
SPB	n	Spring naar n als (B) is positief of ± 0
SNB	n	Spring naar n als (B) is negatief
SOB	n	Spring naar n als (B) is 0

Deze opdrachten hebben steeds betrekking op rekenkundige resultaten in A of B accumulator.

Bij het werken met alfanumerieke informatie kan men natuurlijk niet rekenen en daardoor ook moeilijk spreken over groter of kleiner. Daarom is er een speciale vergelijkingsopdracht.

SAB n Spring naar n als (A) = (B)

Voor deze opdracht worden alle bits van A vergeleken met die van B en alleen indien alle overeenkomstige bits gelijk zijn, wordt de sprong uitgevoerd. Voor deze opdracht is dus + 0 niet gelijk aan - 0. Voor de volgende voorwaardelijke sprongopdracht moeten we eerst nog even terug naar de reeds besproken rekenopdrachten. Bij een opdracht zoals OPA kan het voorkomen dat het resultaat van de optelling groter is dan 11 cijfers. De daardoor

ontstane "overloop", dat is een overdracht naar een niet bestaande 12e cijferplaats, gaat daarbij verloren.

Een zelfde effekt kan optreden bij de opdrachten AFA, OPB en AFB. Bij deze opdrachten wordt de bewerking dus wel uitgevoerd maar het resultaat is foutief.

Een soortgelijke situatie kan zich voordoen bij de opdracht DLN. Wanneer de deler kleiner is dan de kop van het deeltal (dat in A staat) of wanneer de deler gelijk is aan 0 wordt de deling in het geheel niet uitgevoerd. Er bestaan dus verschillende opdrachten waarbij in bijzondere gevallen iets fout kan gaan. Wat er precies bij iedere opdracht gebeurt zal bij de nog te bespreken opdrachten expliciet worden aangegeven.

ALARMREGISTER

Voor alle hiervoor genoemde soort opdrachten geldt echter één gemeenschappelijk ding. In de SERA is aangebracht een zgn. *alarmregister*. Dit alarmregister kent slechts twee toestanden n.l. aan of af. Iedere opdracht nu waarbij iets fout kan gaan beïnvloedt altijd dit alarmregister met dien verstande, dat het wordt afgezet indien de opdracht goed verloopt en wordt aangezet als de opdracht foutief of helemaal niet afloopt. Om het eventuele falen van een bepaalde opdracht ook programmatisch te kunnen onderkennen is er een speciale voorwaardelijke sprongopdracht, waardoor we de stand van dit alarmregister kunnen onderzoeken.

SIA n Spring naar n indien alarmregister is aangezet.

Bij het gebruik van deze opdracht moet men dus bedenken dat het resultaat afhangt van de laatste gebruikte opdracht, die het alarmregister kan beïnvloeden en daarbij laatst bedoeld in dynamische zin dus in de volgorde waarin de opdrachten tijdens het rekenen achter elkaar worden uitgevoerd.

EEN BIJZONDERE VOORWAARDELIJKE OPDRACHT

Tenslotte nog een bijzondere soort voorwaardelijke opdracht.

STP n Stop. Spring naar n als de starttoets van de machine wordt ingedrukt.

Door deze opdracht kan een lopend programma dus onbeperkte tijd worden opgehouden bijv. voor het inleggen van een nieuwe ponsband of een stel ponskaarten of het verwisselen van magneetbanden.
Deze opdracht behoort wel tot de voorwaardelijke opdrachten (het drukken op de startknop) maar het is geen vertakkingsinstruktie omdat nooit wordt doorgegaan met de opdracht vlak achter STP opdracht. Door sprongopdrachten

is het dus mogelijk bepaalde stukken van een programma meerdere keren te laten uitvoeren. Men spreekt dan wel van lussen in een programma.Het aantal malen dat een bepaalde lus dan doorlopen wordt kan op 2 verschillende manieren plaats vinden. Bij de eerste methode wordt het aantal keren van te voren vastgelegd. In de lus zelf moeten dan enige opdrachten voorkomen die een "telling" bijhouden en steeds controleren of de telgrens nog niet bereikt is. Bij de tweede methode ligt het aantal herhalingen niet van te voren vast maar hangt het eindcriterium op een of andere manier van de berekening zelf af.

VOORBEELDEN

Allereerst een berekening met vast aantal keren. Gevraagd wordt de som van de kwadraten van alle getallen 1 t/m 1000. Het rekenproces kunnen we als volgt omschrijven. We beginnen met een som 0. Iedere keer nemen we een volgend getal, berekenen het kwadraat en tellen dat op bij de som. Dit doen we dan 1000 keer, waarna de gehele berekening klaar is. Zoals bij vele problemen moeten we bij het opstellen van het programma rekening houden met de mogelijkheden van de machine wat betreft getalcapaciteit. Het kritieke getal in dit probleem is natuurlijk de som. Past deze som nog in enkele woordlengte in SERA? Nu is het grootste voorkomende kwadraat 1000^2. De gevraagde som is dus zeker kleiner dan 1000 maal dit kwadraat, en daardoor hooguit een getal van 9 cijfers hetgeen "past" in een enkele lengte.

Voor het volgende programma vanaf plaats 100 nemen we aan dat op de plaats 200 het getal 1 en op 201 het getal 1001 staat. Verder gebruiken we de plaatsen 300 en 301 voor het onthouden van de telling en de som.

```
100    HPA   200
101    BSA   300      zet telling op 1
102    BPA   301      maak som = 0
103    HPB   300
104    VMG   300      bereken volgend kwadraat
105    OPB   301
106    BPB   301      bepaal nieuwe som
107    HPB   300
108    OPB   200      telling verhogen
109    BPB   300
110    AFB   201
111    SNB   103      test de telling, indien nog niet
                      klaar voer berekening opnieuw uit
112
.
.
.
200    1              constanten
```

Ir. D.H. Wolbers

300	telling	werkregister
301	som	

Het programma gaat dus weer verder met de opdracht op 112 (hier verder niet aangegeven) als de gevraagde som is berekend en klaar staat in 301. We zien in dit voorbeeld ook dat er een splitsing van het programma in 3 delen optreedt n.l. een instruktiegedeelte, een aantal constantenregisters en een aantal werkregisters. Deze stukken sluiten in het geheugen weliswaar niet mooi op elkaar aan, maar bij het opstellen van een programma kan men in het algemeen moeilijk anders.Dit komt omdat men van te voren niet kan overzien hoe groot ieder stuk zal worden, omgekeerd kan men een programma niet maken wanneer men niet weet waar verschillende gegevens, getallen en resultaten in het geheugen moeten staan. Het geheugen wordt dan echter zeer inefficient gebruikt. Een oplossing zou kunnen zijn om een programma eerst in voorlopige vorm te schrijven zoals in dit voorbeeld. Wanneer men dan weet hoe groot ieder stuk wordt, het gehele programma opnieuw schrijven met een nieuwe nummering en dit als definitief programma gebruiken.
Dit is natuurlijk bijzonder omslachtig. In een der volgende hoofdstukken over symbolische adressen zullen we echter een methode leren kennen waardoor in feite deze tweede fase door de machine zelf m.b.v. het invoerprogramma wordt verzorgd.

Ook voor het volgende voorbeeld plaatsen we weer het programma vanaf 100, de constanten vanaf 200 en de werkruimte vanaf 300. Dit voorbeeld is een onderdeel van een groter programma om een getal in factoren te kunnen ontbinden.
We nemen daartoe het eenvoudige volgende proces.
We delen het getal eerst door 2 daarna door 3 en vervolgens delen we door ieder volgend oneven getal. Het proces van steeds opnieuw delen kan afhankelijk van het getal op een der volgende manieren eindigen. Na een deling vinden we een rest 0. Dan was de laatste deler dus een faktor van dat getal. Het kan ook zijn dat het getal een zogenaamd priemgetal is en alleen maar deelbaar door zichzelf. We behoeven dan met alle oneven delers niet door te gaan tot we het getal bereikt hebben, maar slechts zover tot het quotient kleiner wordt dan de deler. Zou er n.l. wel een grotere deler bestaan waarbij de deling opgaat, dan zou daar een nog kleiner quotient bij horen, maar dan hadden we dit quotient reeds eerder als deler moeten vinden.
We nemen aan dat het getal reeds gegeven is op plaats 300. Na afloop van het proces moet de kleinste deler waarbij de deling opgaat in 302 staan. In het programma wordt de deling door 2 nog afzonderlijk behandeld. Het "lusgedeelte" hier van 105 t/m 113 heeft betrekking op

Ir. D.H. Wolbers

het delen van alle oneven getallen te beginnen met 3. In dit programma komen zowel voorwaardelijke als absolute sprongopdrachten voor. Daarbij is gebruikelijk de eerste te onderstippelen en de laatste te onderstrepen. Let ook op de instruktie op 119 die vanuit de verschillende vertakkingen weer gemeenschappelijk gebruikt wordt.
Het programma gaat altijd weer verder met de hier niet meer aangegeven instruktie op 120 waarbij dan de kleinste gevonden deler waarmee de deling op gaat, als antwoord in 302 is geplaatst.

```
100   HPA   200        0 ⟶ A t.b.v. deling
101   HPB   300        getal ⟶ B
102   DLN   201        deel getal door twee
103   SOA   114        spring als getal deelbaar door 2
104   HPA   202        start met deler 3
105   BSA   301        deler ⟶ 301; 0 ⟶ A t.b.v. volgende
106   HPB   300        deel door volgende          /deling
107   DLN   301        deler
108   SOA   116        spring als rest = 0
109   AFB   301
110   SNB   118        spring als deler groter dan quotient
111   HPA   301        verhoog deler met 2
112   OPA   201
113   SAL   105        hier terug naar begin van de lus
114   HPB   201        kleinste deler is 2
115   SAL   119
116   HPB   301        kleinste deler is de inhoud
117   SAL   119
118   HPB   300        kleinste deler is getal zelf
119   BPB   302        breng antwoord naar 302
120
 ?
200   0
201   2                constanten
202   3
 ?
300   getal
301   deler            werkruimte
302   antwoord
```

Behalve de absolute sprong op 113 die terugspringt naar een reeds eerder opgeschreven opdracht hebben alle andere sprongopdrachten, zowel de voorwaardelijke als de absolute, een andresgedeelte dat verwijst naar een verder gelegen adres, dan het adres waar deze instruktie zelf staat. Bijv. de instruktie op 103 verwijst naar 114. Tijdens het "programmeren", dus het opschrijven van de instrukties laat men deze adresplaatsen aanvankelijk ook open. Is men in dit voorbeeld gekomen aan de instruktie op 113 die weer

terugwijst, dan weet men dat vanaf 114 een andere tak van het programma geplaatst kan worden. Op dat moment vult men adres 114 in de instruktie op 103 pas in. Deze gedragslijn vervolgt men tot er geen "lege" adresplaatsen meer zijn, waardoor dan alle takken dus zeker met elkaar verbonden zijn.

==========
==========

Dit is dan het einde van Uw eerste lesdeel. Een lesdeel dat U waarschijnlijk niet gemakkelijk gevallen zal zijn. Maar weest U ervan overtuigd dat dit ook al gelijk één van de moeilijkste delen uit de kursus was. Indien U deze les goed hebt begrepen zullen de volgende lessen U ongetwijfeld gemakkelijker vallen.

Alvorens U aan het huiswerk begint dient U van Uzelf wel zeker te zijn dat U de voorafgaande stof beheerst. Zo niet doet U er verstandig aan de "zwakke plekken " nog eens door te nemen. Bedenk wel : Alle begin is moeilijk.

Op de volgende pagina's treft U een serie vragen en opgaven aan. De vragen zijn gesplitst in een A- en een B-serie. Wat betreft de A-serie dient U deze allen op te lossen en als huiswerk in te sturen.
Wat betreft de B-serie dient U op bijgevoegd uitwerkpapier slechts die opgaven in te sturen die voorzien zijn van drie plustekens (+++) De overige vragen en opgaven uit Serie-B dient U echter wel voor Uzelf uit te werken.

BIJLAGE

SERA-HEXADECODE

binair	betekenis	kaartcode	sneldrukker		ponsband	L of C
0	0	0	0	4)	101.10	C
1	1	1	1		101.11	C
2	2	2	2		100.11	C
3	3	3	3		000.01	C
4	4	4	4		010.10	C
5	5	5	5		100.00	C
6	6	6	6		101.01	C
7	7	7	7		001.11	C
8	8	8	8		001.10	C
9	9	9	9		110.00	C
10	A	12-1	A		000.11	L
11	B	12-2	B		110.01	L
12	C	12-3	C		011.10	L
13	D	12-4	D		010.01	L
14	E	12-5	E		000.01	L
15	F	12-6	F		011.01	L
16	G	12-7	G		110.10	L
17	H	12-8	H		101.00	L
18	I	12-9	I		001.10	L
19	J	11-1	J		010.11	L
20	K	11-2	K		011.11	L
21	L	11-3	L		100.10	L
22	M	11-4	M		111.00	L
23	N	11-5	N		011.00	L
24	O	11-6	O		110.00	L
25	P	11-7	P		101.10	L
26	Q	11-8	Q		101.11	L
27	R	11-9	R		010.10	L
28	S	0-2	S		001.01	L
29	T	0-3	T		100.00	L
30	U	0-4	U		001.11	L
31	V	0-5	V		111.10	L
32	W	0-6	W		100.11	L
33	X	0-7	X		111.01	L
34	Y	0-8	Y		101.01	L
35	Z	0-9	Z		100.01	L
36		12-7-8			011.01	C
37		11-7-8			110.10	C
38	*	11-4-8	*		110.01	C
39	TWNR	------	NR	1)	000.10	-
40	10	7-8	10		101.00	C
41	negeer	------ 3)	negeer		------	-
42	spatie	0-3-8	spatie		001.00	-
43	tabulator	------	negeer	2)	010.01	C
44	-	11	------		000.11	C
45	=	3-8	=		111.10	C

Opgaven les 1. 001/002

SERIE-A A.1

Naam

Adres

Woonplaats

Kursistennr.

Voor de onderstaande vragen dient U het rondje op te vullen van de antwoorden die volgens U goed zijn. Wij wijzen U er echter bij voorbaat op dat bij een vraag meerdere antwoorden en zelfs alle antwoorden goed kunnen zijn. Het tegenovergestelde kan zich ook voordoen. M.a.w. er kan op een bepaalde vraag ook geen enkel antwoord goed zijn.

Na het oplossen van deze vragen dient U de pagina's van Serie-A in te sturen aan E.C.S. Postbus 2 Heerlen.

Veel succes.

Vraag 1.
- O Informatie is datgene, dat in de computer gaat en er in een andere vorm weer uit komt.
- O Informatiedragers zijn de genomen beslissingen
- O Informatieverwerking is het verwerken van gegevens per computer in deze volgorde : lezen, sorteren, rekenen, onthouden, kontroleren, schrijven.

Vraag 2.
- O Analoog computers zijn goedkoper dan digitale computers.
- O Digitaal computers berusten op het principe van meten.
- O Analoog computers worden zowel voor administratieve als voor wetenschappelijke doeleinden gebruikt.
- O Digitaal computers werken absoluut en zijn uiterst nauwkeurig.

Vraag 3.
- O De SERA computer is een hypothetische machine.
- O De SERA computer werkt zuiver binair.
- O De SERA computer heeft een geheugen met variabele woordlengte. De maximale lengte van een woord is 48 bits.
- O De SERA is een twee adres machine.

Vraag 4.
- O Een woord is de kleinste eenheid van een geheugen.
- O In de SERA-taal heeft ieder woord een adres.

Opgaven les 1

SERIE-A

Naam

Woonplaats

O De inhoud van een woord is steeds gelijk aan z'n adres

O Een SERA-woord is steeds 12 tetrades of 8 hexades lang.

Vraag 5. O De ponskaartlees- en de ponskaartponsmachine zijn invoerorganen.

O Een magnetische bandeenheid is zowel invoer- als uitvoerorgaan.

O Regeldrukker, console en ponsbandponsmachine zijn uitvoereenheden.

Vraag 6. O Een alfanumerieke code bevat alfabetische en numerieke tekens.

O In een alfanumerieke code worden de numerieke cijfers met minder bits weergegeven dan de alfabetische letters.

O Een alfanumerieke code kan ook leestekens en/of speciale tekens (bijv. %, &, +) bevatten.

O Een byte kan 2 cijfers of een letter weergeven.

Vraag 7. O De SERA-geheugen adressering werkt met z.g. vlagbits.

O De SERA-geheugen adressering wordt vergemakkelijkt door de grens-symbolen.

O Geheugen adressering op de SERA-machine gebeurt door begin- en eindadres te speciferen.

Vraag 8. O De excess_three code werd hoofdzakelijk om technische redenen ontworpen.

O De excess-three code bestaat uit eenheden van 3 bits.

O In de 2 uit 5 code bevat iedere pentade steeds 2 0-bits.

O Het pariteitsbit wordt toegevoegd om ook alfanumerieke tekens te kunnen weergeven.

Vraag 9. O Een serie-machine verwerkt de verschillende cijfers tijdens een rekenbewerking in de tijd achter elkaar.

O Een serie-machine is goedkoper en vlugger dan de parallelmachine.

Opgaven les 1 001/002

A.3

SERIE-A

Naam

Woonplaats

0 Een parallelmachine bevat meestal woorden van variabele lengte.

Vraag 10.0 Negatieve getallen hebben in het 10 complement systeem en in het 9 complement systeem dezelfde representatie.

0 Het 1 complement systeem noemt men ook wel eens het "inverse" systeem.

0 In de SERA-code wordt de meest linkse tetrade in een woord als teken beschouwd. + is daarbij 0000 en voor - staat er 1111.

Vraag 11.0 Het centrale geheugen van de SERA-machine wordt gevormd door een magneettrommelgeheugen.

0 Magnetische schijf- en magnetische bandgeheugens duidt men wel aan als random-acces geheugens.

0 De magneetringetjes van een kernengeheugen hebben bijzonder slechte magnetische eigenschappen.

Vraag 12.0 Het rekenorgaan bestaat uit de accumulatoren A en B en het opdrachtenregister.

0 Het rekenorgaan bezit alleen het uitvoeringsregister.

0 Het besturingsorgaan wordt gevormd door de opdrachtenteller en het uitvoeringsregister.

0 In het besturingsorgaan wordt niet gerekend.

Vraag 13.0 Het invoerprogramma leest de informatie-kaarten voor sommige programma's.

0 Het invoerprogramma is in een symbolische taal geschreven.

0 Het invoerprogramma leest en vertaalt de ingevoerde programmakaarten.

0 Het invoerprogramma moet door de programmeur zelf worden geprogrammeerd.

Vraag 14.0 Drijvende komma representatie wordt vooral in de wetenschappelijke sfeer toegepast.

0 Voor drijvende komma getallen moeten steeds 2 woorden in het geheugen worden gereserveerd.

0 Drijvende komma getallen zijn veel nauwkeuriger dan vaste komma getallen.

Opgaven les 1

Naam

SERIE-A

Woonplaats

Vraag 15. O Voor een machine die genormaliseerde drijvende komma getallen hanteert hebben de grootheden a en b ($g = ax10^b$) respectievelijk de volgende waarden :
a is max. 6
b is max. 2
Voor welke onderstaande drijvende komma getallen zal er een foutmelding optreden.

O + 0,920000 + 02

O - 0,036700 + 04

O + 0,250000 - 1,5

O - 0,112233 - 99

O + 0,1236543 + 06

O + 0,652821 + 103

Vraag 16. O Een SERA-instruktie wordt voor uitvoering geplaatst in :

O alarmregister

O accu-A

O opdrachtenteller

O accu-B

Vraag 17. O Haal en brenginstrukties verzorgen invoer en uitvoer van gegevens.

O Haalinstrukties vernietigen de vorige inhoud van de registers A of B.

O Brenginstrukties vernietigen de vorige inhoud van de registers A of B.

Vraag 18. O Na een SERA "deel" opdracht bevinden zich het quotient en de rest respectievelijk in A en B.

O Na een SERA "deel" opdracht bevinden zich het quotient en de rest respectievelijk in B en A.

O Na een SERA vermenigvuldiging zijn de vorige inhouden van A en B beide overschreven.

Vraag 19. O Een spronginstruktie wordt in een programma alleen ingelast om over konstanten en werkregisters te springen.

O De meest voorkomende SERA spronginstrukties zijn onvoorwaardelijke spronginstrukties.

Opgaven les 1

Naam

SERIE-A

Woonplaats

O De STP-instruktie is een voorwaardelijke sprong-instruktie.

O Een lus in een programma is ondenkbaar zonder spronginstrukties.

Vraag 20. O Een SIA-instruktie zet het alarmregister aan.

O SIA is een voorwaardelijke spronginstruktie.

O SIA mag men per programma maar een keer gebruiken.

Vraag 21. O Een programma is een serie instrukties, in sequentiële volgorde, waartussen een logische relatie bestaat.

O Een programma is het resultaat van de omzetting van een probleem in een bepaalde computertaal.

O Een SERA-programma wordt in ponskaarten geponst, die door het invoerprogramma worden ingelezen en vertaald, waardoor er een uitvoerbaar programma in machinecode in het geheugen komt te staan.

Dit waren de vragen van Serie-A. Alvorens U de antwoorden instuurt, kontroleert U even of op elk blad Uw naam, enz. is ingevuld.
Dank U.

Opgaven les 1 001/002

B.1

SERIE-B

Onderstaande vragen en opgaven dient U allen voor Uzelf uit te werken. De opgaven voorzien van drie kruisjes (+++) dient U echter uit te werken op het bijgevoegde uitwerkpapier en in te sturen ter beoordeling aan E.C.S. Postbus 2 Heerlen.

Vraag 1.a. Welke 5 belangrijkste organen kan men aan een +++ computerinstallatie onderkennen?

b. Geef schematisch het onderlinge informatietransport weer tussen deze 5 componenten.

c. Definieer kort de functie van ieder orgaan.

Vraag 2.a. Omschrijf minstens drie bekende geheugenvormen.

b. Wat noemt men de toegangstijd van een geheugen?

Vraag 3.a. Wat zijn andere namen voor resp. het 2-, 8-, +++ 10-, en 16-tallig stelsel?

b. Geef van elk dezer talstelsels de getallen (in oplopende volgorde), die nog met 1 teken geschreven kunnen worden.

c. Geef een definitie voor het grondtal (basis) van een willekeurig talstelsel.

Vraag 4. Schrijf voluit (als de som van een aantal machten +++ van 10) de getallen :

a. $(172653)_{10}$ dus $3x10^0$, $5x10^1$, enz.

b. $(172,653)_{10}$

c. $(0,00172653)_{10}$

Vraag 5. In welke 2 fasen verloopt de konversie van een gebroken decimaal getal naar een ander talstelsel?

Vraag 6. Konverteer naar het 8-tallige stelsel :

a. $(13579)_{10}$

b. $(3,125)_{10}$

c. $(4,0673)_{10}$, antwoord afronden op 3 octalen achter de komma.

d. Waarom is de opmerking onder c niet nodig bij b.

Vraag 7. Konverteer naar het 16-tallig stelsel :

+++ a. $(64890)_{10}$
b. $(32,0625)_{10}$
c. $(0,073)_{10}$, in twee 16-tallige cijfers achter de komma.

Vraag 8. Konverteer naar het 2-tallig stelsel :

+++ a. $(2)_{10}$
b. $(31)_{10}$
c. $(123,5)_{10}$
d. $(18,47)_{10}$, nauwkeurig tot 5 bits achter de komma.

Vraag 9. Konverteer naar het 10-tallig stelsel (eventueel op 4 decimalen na de komma) :

a. $(2460)_7$
b. $(1101001101)_2$
c. $(101101)_8$
d. $(11)_{16}$
e. $(AD3F)_{16}$
f. $(24,43)_7$
g. $(11,0101)_2$
h. $(6,1)_8$
i. $(634,004)_8$
j. $(F,F3)_{16}$

Vraag 10. Konverteer $(876)_{10}$ naar het 2-tallig stelsel en vorm hieruit, zonder gebruikmaking van het decimale stelsel het 4-, 8-, en 16-tallige equivalent.

Vraag 11. Evenals in het 10-tallig stelsel kan men in een
+++ ander stelsel tafels van vermenigvuldiging opstellen. Geef de tafel van 6 in het 16-tallig stelsel dus :
1x6 =
2x6 =
...
...
9x6 =
Ax6 =
...
...
Fx6 =

Vraag 12.a. Vorm de som en het produkt van de binaire getallen 1011101 en 100011110.
b. Controleer de uitkomsten door de twee getallen te converteren naar het decimale stelsel, de rekenkundige bewerkingen als zodanig uit te voeren en de resultaten te vergelijken met de tientallige voorstelling van de eerder gevonden antwoorden.

SERIE-B

c. Geef de binaire rekenregels voor optellen en vermenigvuldigen.

Vraag 13.a. Een getal in drijvende komma (floating point) notatie bestaat uit 2 delen. Hoe heten deze?
+++

b. Wat betekent normaliseren in dit verband?

c. Hoe kan schijnnauwkeurigheid (zg. underflow) ontstaan?

Vraag 14.a. Geef voor de waarde $(530)_{10}$ zowel de binaire als de decimaal/binaire weergave.

b. Verklaar waarom er voor de decimaal/binaire voorstelling meer bits nodig zijn.

Vraag 15. Geef de binaire voorstelling van het decimale getal 8061 in de excess-three code (incl. het pariteitsbit, dus ieder cijfer als een pentade weergeven)
+++

Vraag 16. Hoe zullen de volgende decimale waarden er in een compleet SERA-woord uitzien.
+++

a. 573

b. +13

c. -736895

d. -0

Vraag 17. Geef het resultaat (quotient en rest) van de volgende delingen m.b.v. de SERA-machine.

a. 500/10

b. -480/25

c. -1/-5

d. 80/-12

Vraag 18. Voor de volgende oefeningen geldt:
+++
adres van p = 1201
adres van q = 1202
adres van r = 1203
adres van s = 1204
adres van t = 1205
adres van x = 1301
adres van y = 1302
De eerste instruktie moet staan op adres 1000, ieder (tussen)resultaat past in 1 woord. Geef het programmadeel voor de berekening van :

a. x=p+q-r+s-t

b. x=pq-r en tegelijk y=pq

c. x=pqr-s en y=o

d. breng p t/m t via een programma verder in het geheugen

e. eveneens als d doch na de verhuizing moeten er nullen staan op de oude adressen p t/m t.

Vraag 19. Vergelijk opgave 18. Geef het programmadeel voor de berekening van :
y=r/s-t en
x=p+ de rest van de deling r/s als gegeven is dat q=o.

Vraag 20. Vergelijk opgave 18. Geef het programmadeel voor de berekening van :
x=pq+r(s+tb)-t
y=(s+tb)pr
Gebruik 1401 en volgende voor eventuele tussenresultaten.N.B. Vermijd uiteraard zoveel mogelijk dubbel werk. Ieder (tussen) resultaat past in 1 woord. Er is geen adres bekend waarvan de inhoud nul is.

Vraag 21. Als opgave 18, doch nu is van geen enkel (tussen) resultaat zeker of het nog wel in 1 woord past, 2 woorden zijn dan echter voldoende.Kies daarom voor x en y ieder 2 adressen voor de kop (die nul kan zijn) en de staart, gebruik bovendien steeds 2 adressen voor de opslag van een tussenresultaat.

Vraag 22. Maak een programma voor het volgende probleem. Beschouw de getallenreeks van 1 t/m 250. Bepaal de 4 sommen van alle getallen uit die reeks, die aan de volgende voorwaarden voldoen :

a.deelbaar zijn door 2 (resultaat op adres 2000)

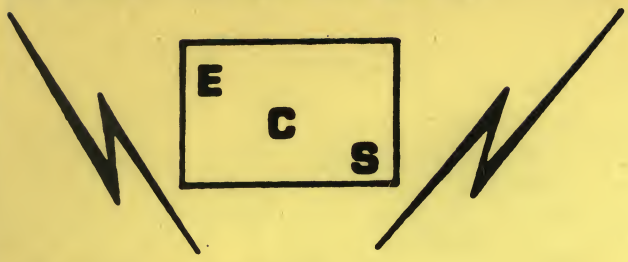
b.deelbaar zijn door 7 (resultaat op adres 2001)

c.deelbaar zijn door 8 (resultaat op adres 2002)

d.niet deelbaar zijn door 2 en 7 (resultaat op adres 2003)

Kies zelf adressen voor instrukties, konstanten en eventueel tussenresultaten. Alle (tussen) resultaten passen in 1 woord. Beredeneer waarom de som van de velden 2000 t/m 2003 niet gelijk is aan de som van alle getallen uit de reeks 1 t/m 250.

european computer school

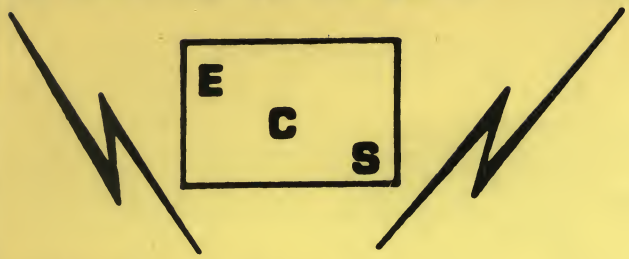


PASSAGE 4-6. HEERLEN

Naam :
Adres :
Woonplaats :
Kursistennr. :

U I T W E R K P A P I E R

european computer school

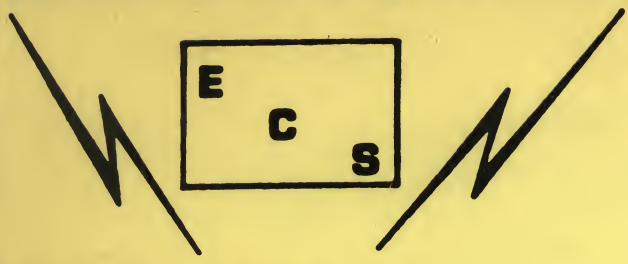


PASSAGE 4-6. HEERLEN

Naam :
Adres :
Woonplaats :
Kursistennr. :

U I T W E R K P A P I E R

european computer school



PASSAGE 4-6, HEERLEN

Naam :
Adres :
Woonplaats :
Kursistennr. :

U I T W E R K P A P I E R